

HERIOT-WATT UNIVERSITY

Real-Time People Tracking in a Camera Network

by

Wasit Limprasert

Submitted for the degree of Doctor of Philosophy

Heriot-Watt University

School of Mathematical and Computer Sciences

November 2012

The copyright in this thesis is owned by the author. Any quotation from the thesis or use of any of the information contained in it must acknowledge this thesis as the source of the quotation or information.

HERIOT-WATT UNIVERSITY

Abstract

Heriot-Watt University
School of Mathematical and Computer Sciences

Doctor of Philosophy

by Wasit Limprasert

Visual tracking is a fundamental key to the recognition and analysis of human behaviour. In this thesis we present an approach to track several subjects using multiple cameras in real time. The tracking framework employs a numerical Bayesian estimator, also known as a particle filter, which has been developed for parallel implementation on a Graphics Processing Unit (GPU). In order to integrate multiple cameras into a single tracking unit we represent the human body by a parametric ellipsoid in a 3D world. The elliptical boundary can be projected rapidly, several hundred times per subject per frame, onto any image for comparison with the image data within a likelihood model. Adding variables to encode visibility and persistence into the state vector, we tackle the problems of distraction and short-period occlusion. However, subjects may also disappear for longer periods due to blind spots between cameras fields of view. To recognise a desired subject after such a long-period, we add coloured texture to the ellipsoid surface, which is learnt and retained during the tracking process. This texture signature improves the recall rate from 60% to 70-80% when compared to state only data association. Compared to a standard Central Processing Unit (CPU) implementation, there is a significant speed-up ratio.

To Mom, Dad and Wa

ACADEMIC REGISTRY

Research Thesis Submission



Name:	WASIT LIMPRASERT		
School/PGI:	Mathematical & Computer Science		
Version: <i>(i.e. First, Resubmission, Final)</i>	Final	Degree Sought (Award and Subject area)	PhD in Computer Science

Declaration

In accordance with the appropriate regulations I hereby submit my thesis and I declare that:

- 1) the thesis embodies the results of my own work and has been composed by myself
- 2) where appropriate, I have made acknowledgement of the work of others and have made reference to work carried out in collaboration with other persons
- 3) the thesis is the correct version of the thesis for submission and is the same version as any electronic versions submitted*.
- 4) my thesis for the award referred to, deposited in the Heriot-Watt University Library, should be made available for loan or photocopying and be available via the Institutional Repository, subject to such conditions as the Librarian may require
- 5) I understand that as a student of the University I am required to abide by the Regulations of the University and to conform to its discipline.

* Please note that it is the responsibility of the candidate to ensure that the correct version of the thesis is submitted.

Signature of Candidate:		Date:	
-------------------------	--	-------	--

Submission

Submitted By <i>(name in capitals)</i> :	
Signature of Individual Submitting:	
Date Submitted:	

For Completion in the Student Service Centre (SSC)

Received in the SSC by <i>(name in capitals)</i> :			
Method of Submission <i>(Handed in to SSC; posted through internal/external mail):</i>			
E-thesis Submitted (mandatory for final theses)			
Signature:		Date:	

Acknowledgements

I would like to express my very great appreciation to Professor Andrew Wallace and Professor Greg Michaelson, my research supervisors, for their patient guidance, enthusiastic encouragement and useful critiques of this research work.

I wish to thank various people for their contribution to this project; Dr Manuel Martinello, Dr Claire Morand, Ms Eleonora Arca, Mr Komsan Kanjanasit, Ms Sirisuda Buatongkue and Ms Sushma Bomma, for their valuable support in the dataset acquisition; Dr Paolo Favaro for allowing me to use his cameras in order to obtain the DEC11 dataset. I am particularly grateful for the intrinsic calibration crosschecking given by Mr Daniele Perrone. I would also like to extend my thanks to the technicians of the EPS workshop for their help in camera installation and IT staff of the MACS school for software resources.

Finally, I wish to thank my parents and my wife for their support and encouragement throughout my study.

Contents

Abstract	i
Acknowledgements	iv
List of Figures	x
List of Tables	xiii
Symbols	xiv
1 Introduction	1
1.1 Problems and specifications	3
Distraction	4
Short time disappearance	4
Speed	4
Scalability of the camera network	4
1.2 Thesis structure	4
2 Background Theory	6
2.1 Tracking	6
2.1.1 Similarity and distance	6
2.1.2 Prediction	7
2.2 Review on Visual Tracking	9
2.2.1 Appearance Descriptors	11
2.2.1.1 Boundary and Region	12
Contour	13
Polygon	13
Parametric shape	13
Region and Silhouette	14
2.2.1.2 Spatial pattern	14
Optical flow	14
Saliency	14
Stable feature points	14
Wavelet basis	15
HOG	15

	LBP	15
	HOG+LBP	15
	Gabor feature	15
2.2.1.3	Adaptive image	16
	Database of images	16
	WSL image	17
2.2.1.4	Colour	17
2.2.1.5	3D model	18
	Vertices	18
	Visual hull	18
2.2.2	State Estimators	18
2.2.2.1	Maximum-Likelihood	19
2.2.2.2	Maximum a Posterior	20
2.2.2.3	Bayesian Estimator	21
2.2.2.4	Data Association	21
	Probability data association	21
	Shortest path	22
	Recognition	23
2.2.2.5	Detection	23
	Background Segmentation	23
	People detection	24
2.3	Mean-Shift method	24
2.4	Kalman Filter	27
2.5	Numerical Bayesian estimator	31
2.5.1	Monte Carlo methods	34
2.5.1.1	Importance sampling	35
2.5.1.2	Rejection sampling	36
2.5.1.3	Inversion sampling	37
2.5.2	Markov Chain Monte Carlo methods	38
	Drawbacks of MCMC	39
	Parallel MCMC	40
2.5.3	SIR Particle filter	40
	State variables	41
	Transition functions	41
	Prior density	41
	Likelihood function	42
	Posterior density	42
	Re-sampling	42
2.5.4	Likelihood and Similarity measure	44
2.5.5	Summary	45
2.6	Camera Calibration	45
2.6.1	Homography	49
2.6.2	Camera calibration method	51
2.7	Hardware comparison	53
2.8	Summary	55
2.8.1	Research direction	58

3	Development of Multi-Target Multi-Camera tracking	60
3.1	Single person tracking	61
3.1.1	SIR Particle filter	62
	Data-structure	63
	Likelihood function	63
	Resampling	64
	Transition function	65
3.1.2	Basic similarity measure	65
3.1.3	Experiment and result	69
3.2	Fast Ellipsoid projection	72
3.2.1	Ellipsoid projection	72
3.2.2	Texture mapping	77
3.2.3	Summary	80
3.3	Ellipsoid Likelihood	80
	Vertex base ray tracing:	80
	Parametric ellipsoid ray tracing:	81
3.3.1	Silhouette similarity	82
3.3.2	Ownership of a pixel	85
3.3.3	Texture learning and texture likelihood	86
3.4	Our Particle Filter for Multiple Target Tracking	90
3.4.1	Likelihood with distraction suppression	90
3.4.2	Visibility state	91
3.5	Summary	92
4	Sequential Implementation	94
4.1	Multiple Target Detection and Tracking Framework	95
4.1.1	Detection	95
4.1.2	Tracking	97
4.2	Sequential Algorithm	99
4.2.1	Implementation	100
4.3	Experiment on Sequential Implementation	102
4.3.1	Advantage of using texture similarity	103
4.3.2	Accuracy evaluation	105
4.3.3	Computing time	110
4.4	Comparison	112
4.5	Summary	114
5	Parallel Implementation	117
5.1	Parallel Design	118
5.1.1	GPU architecture	119
	Global memory	121
	Constant memory	121
	Shared memory	121
	Texture memory	121
	Register	122
	CUDA programming	122
5.1.2	Parallelism methods	123

5.1.2.1	Data Parallelism	124
5.1.2.2	Reduction	127
5.1.2.3	Skip ahead	128
5.1.2.4	Use of fast memory	129
5.2	Implementation	130
5.3	Speed evaluation	131
5.4	Conclusion	136
6	Tracking in a disjoint camera network	138
6.1	Related work	139
6.2	Our approach	141
6.3	Theory	143
6.3.1	Assignment problem	144
6.3.2	Spatial association probability	145
6.3.3	Texture association probability	146
6.4	Design of Experiments	148
	Recall rate	148
6.4.1	Experiment 1: spatial-only	149
6.4.2	Experiment 2: texture-only	150
6.4.3	Experiment 3: combining spatial and texture	150
6.5	Evaluation	150
6.5.1	Evaluation by using PETS09	151
	Tracking	151
	Data Association	151
	Measure recall rate	152
	Output from data association	152
6.5.2	Evaluation by using DEC11 dataset	155
6.6	Result	155
6.6.1	Result 1: spatial-only data association	156
6.6.2	Result 2: texture-only data association	157
6.6.3	Result 3: Spatial-texture data association	157
6.7	Recall Precision and F-measure	160
6.8	Discussion	163
7	Conclusion	165
	Contribution 1: Likelihood from ellipsoid projection	165
	Contribution 2: Analysis of the tracking framework	166
	Contribution 3: GPU acceleration	166
	Contribution 4: Spatial-Texture data association	167
7.1	Future work	167
	Pan-tilt-zoom cameras	167
	Appearance descriptors	168
	Optimising the GPU implementation	168
	Adaptive combining factor for the spatial-texture cost func- tion	168

A LM Optimization	169
A.1 LM Optimization	169
STEEPEST DESCENT METHOD	170
GAUSS-NEWTON METHOD	170
LM METHOD	171
B Appendix DEC11 dataset	174
B.1 Generating Landmarks	174
B.2 Refining coordinates	175
B.3 Extrinsic Calibration	179
B.4 Calibration parameters DEC11	186
Camera1	186
Camera2	186
Camera3	187
Camera4	187
Camera5	187
Camera6	188
C EM330 dataset	189
C.1 Hardware and software requirements	190
C.2 Calibration parameters EM330	190
Camera1	190
Camera2	191
Camera3	191
D Distance and similarity	193
Bibliography	194

List of Figures

2.1	Tracking system consists of the appearance descriptor and the state estimator.	9
2.2	Components of visual tracking.	12
2.3	Mean-shift seeking density peaks. In this example, uniform kernels are applied to perform gradient ascending over density function resulting in two stationary points at final step.	25
2.4	An approximation of $\int_{x=0}^{x=1} \sqrt{1-x^2} dx$ by MC integration method	35
2.5	Generating an sample by Rejection sampling method	37
2.6	Inversion sampling transforms a uniform distribution to the objective distribution $f(x)$	38
2.7	Three steps of SIR particle filter algorithm from top to bottom: likelihood evaluation, re-sampling and dynamics transition.	43
2.8	A pinhole camera model	46
2.9	Projection from a 3D point (q_x, q_y, q_z) to a 2D point (p_u, p_v) on an image plane at $z = 1$. Note that translation and rotation are removed to simplify the model.	47
2.10	Relation between 3D coordinate system (x,y,z) and pixel coordinate system (u,v).	49
3.1	SIR particle filter consists of 3 functions likelihood, re-sample and transition.	62
3.2	Data structure of the state S	63
3.3	Chi-square distribution (blue) and its cumulative function (green)	66
3.4	Cross-section of cylindrical model.	67
3.5	A rectangle on bottom-left is an example projection of the cylindrical cross-section model.	67
3.6	Black solid line is ground truth and red markers are the estimated trajectory in meter.	70
3.7	From top-left to bottom-right, tracking results of frame 290, 300, 310, 320, 330, 340, 350 and 360. Particles on ground floor show estimated position where green is high weight. The tracker was distracted in the last two frames.	71
3.8	A top-view of the projection system, eye-point $\hat{\mathbf{e}}$, position of ellipsoid $\hat{\mathbf{e}}$ and the direction vector $\hat{\mathbf{d}}$	79
3.9	Transforming Euclidean coordinate to Cylindrical coordinate.	79
3.10	Ray tracing of an ellipsoid in 2 cameras system.	81
3.11	Ellipse Contour.	83
3.12	An ellipse kernel projected on a foreground image. Red area is positive contour, $0 \leq \psi$, and blue region is negative contour, $-0.5 < \psi < 0$	84
3.13	Data structure of the multiple target state S	87

3.14	Markov Chain Model of visibility state of a subject	92
4.1	Detection grids (yellow circles) and ellipsoids projection.	95
4.2	Detection function. The rectangles and the round shapes are data and operations, respectively	97
4.3	Multiple targets tracking framework. The rectangles and the round shapes are stored data and operation, respectively	98
4.4	Background segmentation and particles on ground floor.	100
4.5	A sequential algorithm of the detection tracking system.	101
4.6	A class diagram of the detection tracking programm.	101
4.7	Events in MOTA , miss detection, false alarm and switch events.	103
4.8	Evaluating with PET09 dataset. First and second columns are result sequences from camera1 and camera3, from top to bottom are frame 700th, 720th and 740th. The numbers over ellipse show ID and height in unit meter.	105
4.9	From top-left to bottom-right are frames 135th to 175th from PETS09.	108
4.10	From top-left to bottom-right are frames 1500th to 1560th from PETS06.	108
4.11	From top-left to bottom-right are frames 1500th to 1560th from PETS03.	109
4.12	From top-left to bottom-right are frames 510th to 540th from EM330.	109
4.13	Relation between likelihood computing time and number of particles.	111
4.14	Likelihood Latency time.	116
5.1	CPU to GPU connection and internal connection inside an example GPU (no practical GPU has 2 MPs).	120
5.2	The logical thread-blocks are mapped to available 2 MPs.	123
5.3	Compute summation by the Reduction method.	127
5.4	(left) a series of x_n are generated a sequential cascade process. (right) skip ahead is applied to divide the series into shorter sequences	129
5.5	Sequential diagram of a GPU implementation.	131
5.6	Computation time of all kernel functions.	133
5.7	Memory utilisation of all kernel functions.	134
6.1	Coverage scalability of the joint and the disjoint network.	139
6.2	Transition of Subjects in a disjoint camera network	140
6.3	The data association problem; how do we link the detections with the trajectories?	143
6.4	Texture matching in a video sequence, the trajectory line is broken at step 4.	144
6.5	Recall rate evaluation.	148
6.6	Spatial-texture data association with PETS09 dataset, detail in text	153
6.7	Data association with DEC11 dataset. These samples are taken from the <i>experiment2</i> with $\nu^t = 2 \times 10^{-2}$	154
6.8	Subjects and their labels in the PETS09 sequence.	155
6.9	Subjects and their labels in the DEC11 sequence.	155
6.10	Results of the spatial-texture data association PETS09	158
6.11	Recall rate of the spatial-texture data association DEC11	159
6.12	ROC computed from PETS09.	162
6.13	Characteristic of our data association.	162

B.1	(top) Landmarks and (bottom) Camera views.	176
B.2	Overlay of the landmark coordinates on a Google map.	177
B.3	Iteration process to refine the landmark coordinates.	178
B.4	Projection of Camera1, (top) solution by using homography (middle) by LM optimisation and (bottom) position of camera.	180
B.5	Projection of Camera2, (top) solution by using homography (middle) by LM optimisation and (bottom) position of camera.	181
B.6	Projection of Camera3, (top) solution by using homography (middle) by LM optimisation and (bottom) position of camera.	182
B.7	Projection of Camera4, (top) solution by using homography (middle) by LM optimisation and (bottom) position of camera.	183
B.8	Projection of Camera5, (top) solution by using homography (middle) by LM optimisation and (bottom) position of camera.	184
B.9	Projection of Camera6, (top) solution by using homography (middle) by LM optimisation and (bottom) position of camera.	185
C.1	Camera layout in EM330	189
C.2	Top to bottom are sample images from camera1, camera2 and camera3 . .	192

List of Tables

2.1	The resource requirement to implement floating point operations	54
2.2	CPU GPU and FPGA comparisons	55
2.3	appearance descriptor comparison	56
2.4	MOTA comparison with results in [153]	59
3.1	Mathematical operation requirements for intersection calculation per ob- ject per p pixels	82
3.2	Comparison of the total number of calls in a unit of million, where $n=2$, $v=4$ and $p=100,000$	82
4.1	Comparing using and not using the texture similarity	104
4.2	Error and number of particles.	106
4.3	Profiling of sequential implementation to measure computational time in millisecond evaluated from various datasets	113
4.4	MOTA comparison with results in [153]	114
5.1	Characteristic of device memory of GTS250.	120
5.2	Comparing computation time between sequential and parallel processing.	136
6.1	Selected methods of disjoint camera tracking	141
6.2	Recall rate of the spatial-only data association (PETS09)	156
6.3	Recall rate of the spatial-only data association (DEC11)	156
6.4	Recall rate of the texture-only data association (PETS09 and DEC11)	157
6.5	Recall rate of the 2nd combining method (PETS09)	160
6.6	Recall rate of the 2nd combining method (DEC11)	160
A.1	Steepest descent algorithm	170
A.2	Gauss-Newton algorithm	170
A.3	LM algorithm	171
D.1	List of distance and similarity formulas	193

Symbols

Chapter2

$\langle x \rangle$	A expectation value of x
$\text{var} \langle x \rangle$	A variance of x
$x \sim f()$	A drawing a sample x from a density function $f()$
$\mathcal{N}(\mu, \sigma)$	A normal distribution function with a mean μ and a variance σ^2
$\mathcal{U}(0, 1)$	A uniform distribution function with range from 0 to 1
$\mathcal{P}(S)$	A prior probability density of the state S
$\mathcal{Q}(S X)$	A posterior probability density of the state S confirmed by data X
$\mathcal{L}(X S)$	A likelihood function of data X given the state S
u	Horizontal direction of a image coordinate system (left to right)
v	Vertical direction of a image coordinate system (top to bottom)
K	A 3-by-3 intrinsic matrix in a camera model
R	A 3-by-3 rotation matrix in a camera model
t	A translation vector in a camera model
P	A 3-by-4 camera projection matrix
H	A homography matrix
α	A focal length in horizontal pixel units
β	A focal length in vertical pixel units
γ	Skewness
ϵ_u	A horizontal camera centre on the image plane
ϵ_v	A vertical camera centre on the image plane
p	A pixel coordinate $\mathbf{p} = (p_u, p_v)$
q	A 3D world coordinate $\mathbf{q} = (q_x, q_y, q_z)$

Chapter3

n	A particle index
-----	------------------

t	A time (frame) index
S_n	The state vector of the n^{th} particle
w_n	A weight of the n^{th} particle computed from the likelihood function
\mathcal{W}	Wiener process $\mathcal{W} = \mathcal{W}(\Delta t) \sim \mathcal{N}(0, \sqrt{\Delta t})$
σ	A variance
χ^2	Chi-square
\mathbf{M}	A 3D vertex model
\mathbf{m}	Projected coordinates of \mathbf{M}
\mathbf{e}	An eye-point of a camera
\mathbf{d}	A direction vector in ray-tracing method
τ	A distance from the eye-point to an ellipsoid surface
ψ	A contour value determined from an elliptical contour function
A, B, C	Coefficients of an ellipse function
c_u, c_v	A coordinate of an ellipse centre
$T_{z,\theta}$	Texture-pixel colour at coordinate (z, θ)
Γ^a	A positive silhouette likelihood score
Γ^b	A negative silhouette likelihood score
Γ^c	A texture likelihood score
ν	A threshold of the ramp function in the cost function
ξ	Blending factor in the cost function

List of Publications

- Wasit Limprasert, Andrew M. Wallace, Greg Michaelson: Accelerated People Tracking using Texture in a Camera Network. VISAPP (2) 2012: 225-234

Chapter 1

Introduction

Visual information from our eyes is a very important observation that goes to our brain for planning. Visual perception has a major influence on our intelligent evolution. Adult humans perform visual tracking in almost all tasks and visual tracking is a key for higher level processes such as navigating, learning and recognition. Visual tracking allows human to reduce very rich information from the surrounding environment and mentally process only sufficient useful information. Humans are able to perform face tracking at a very early age because it is so important for survival. In most activities we need to perceive objects around us and we need to concentrate on a single object, for example reading the bus number on a moving bus, playing any ball game, hunting, and so many more.

Visual tracking is a fundamental element in navigation and spatial intelligence. Around 1983, Jean Piaget, a psychologist, studied cognitive development in children[1]. He suggested that an infant acquires visual tracking ability when around one year-old. When two years-old, they have the ability to understand that a disappeared object continues to exist; the ability is called *object permanence*. Object permanence indicates that infants are able to recognise an object for a short period and also able to estimate location, although the object disappears. This ability is the simplest form of tracking developed by experience. Piaget believed the ability to perform tracking was created by learning mechanisms and stimuli such as moving objects. Even the origin of our perception is unclear but it is definitely clear that tracking ability plays a vital role in survival and brings

about a higher level of intelligence for planning and interacting with the surrounding environment.

In a human eye, the fovea [2] is a very small region in the eye containing *cone cells* (photo sensitive cells) that are responsible for colour vision. *The rod cells* work best in dim light and capture gray-scale images. In the fovea all *cone cells* are concentrated in a small area[3], and so the eye captures colour image very sharp at middle. In order to retrieve rich detail from the surrounding scene, the eyes move from point to point quickly to collect information of the scene. This is known as *saccadic eye movement*[4]. In order to recreate perception of a detailed scene, those observations must be mentally merged into a single world. In order to acquire information from a moving subject, the eyes must follow the subject in a short period of time, which is called *smooth pursuit movement*[5] allowing the brain to extract features from a moving subject in a distracting environment. This tracking ability is central to our perception in a dynamic environment.

In machine learning, tracking is a necessary ability for classification and learning. Classification methods require an input feature vector, which is a constant dimension (number of elements). However, the input from a camera is huge and always has missing data due to occlusion. A tracking system has to concentrate on particular subjects. It needs segmentation and localisation methods to separate a considered subject from the surrounding objects. For example, we can use a histogram as a feature vector for classification. Computing the histogram of an entire image is quite easy. However, calculating the histogram of a single person in a crowded scene is relatively much more difficult because the bounding area of the person must be estimated first. In this case, a tracking program can give the bounding box of the person. Alternatively, a tracking system can perform segmentation by manipulating an estimated bounding contour such as active contour [6, 7].

In surveillance, people tracking systems are deployed for behaviour analysis or detecting anti-social behaviour. The activity of a person could be identified by a history of locations in a time series. For example, tracking is applied to detect abandoned bag in [8]. Tracking also can be applied to capture human joint motion in order to create realistic motion in computer graphics. These are a few examples from many possibilities for applying visual tracking for obtaining and interpreting the surrounding environment.

To be precise, visual tracking is performed to find the position or the boundary of subjects by extracting information from a sequence of images captured by cameras. The outputs of a tracking system are bounding boxes or estimated positions corresponding to real subjects. In this thesis, we will show an approach to create multi-target tracking and recognition using probabilistic inference.

In addition we have applied parallel processing to accelerate the tracking process. Normally a tracking module must be integrated with consequence decision and machine control modules to create responsible interaction between the machine system and real world. Therefore the tracking module has to perform quickly to minimize delay for an instant response. Considering the accuracy of a real-time tracking system, a system with low frame rate has a large time-gap between observations. The time-gap brings about uncertainty due to the lack of observation within the gap. So the tracking accuracy is influenced by the frame rate of the observations. Increasing the frame-rate results in reduced uncertainty from the observation.

One promising technology is to exploit a graphics processing unit (GPU) in a real-time tracking application. To compare a GPU with CPU, a mid range GPU model can deliver a huge number of GFLOPS (billion floating point operation per second). For example the NVIDIA GeForce GTS250, which we will use in this thesis, can theoretically reach 705 GFLOPS (128cores) at 4.9Watt per GFLOPS [9], whereas the CPU Intel Corei7 965 (4cores) can perform at 69 GFLOPS at 5.3Watt per GFLOPS [10]. In terms of the computational performance parallel computing in a GPU overtakes the multi-core CPU. Concerning power efficiency, the CPU also consumes more power per GFLOP than the GPU because the power dissipation of a processing unit increases linearly with the clock frequency. In the GPU there are many hundreds of cores running at 1.6 GHz, whilst the CPU runs at about twice speed and at higher power. Hence, the GPU is a good candidate for our tracking implementation in both the speed and power efficiency aspects.

1.1 Problems and specifications

In this thesis, we consider and try to solve four major problem in visual tracking. We use the camera video in order to estimate the position of subjects and recreate the complete

trajectories of all subjects.

Distraction is a major problem in multiple targets tracking. The tracking system must be able to distinguish individual subjects and estimate individual trajectory.

Short time disappearance is absence of observation due to obstruction between the subject and the camera or the mutual occlusion between subjects. Specifically, the subject cannot be seen from a camera for a few seconds (less than 5 seconds). This can be tackled by multiple cameras and a probabilistic model.

Speed is an important factor. The minimum frame rate of CCTV and industrial camera is 7.5 fps (frame per second). The system should operate faster than or equal to this speed.

Scalability of the camera network is important for the real application, where the covered area is very large. An overlapping camera network, where all subjects must be visible to all cameras, has the drawback that the coverage area is limited. In contrast, a disjoint camera network can cover a large area. The coverage is proportional to the number of cameras.

1.2 Thesis structure

In Chapter 2, we review recent visual tracking methods and the hardware for our prototype implementation. We also compare and justify these methods in order to design a real-time tracking prototype.

In Chapter 3, the development of tracking theory and design will be explained. This includes a proposed parametric ellipsoid model, which is theoretically faster than the traditional vertex base by about 10 times because of the simplicity of pixel-wise calculation. We compare our parametric ray-tracking approach with the standard vertex base and show speed improvement analytically. We also design the likelihood model to tackle distraction and short time disappearance.

In Chapter 4, we design and evaluate the tracking algorithm. The implementation of the tracking framework on a CPU is evaluated with various dataset. The computational time of sub-functions is also measured for further comparison.

In Chapter 5, the tracking framework is transferred from a CPU to a GPU and measured speed performance. We show speed improvement of around 4 times compared to the CPU implementation.

In Chapter 6, we also consider data association in order to link broken trajectories, where subjects leave and re-enter the observation areas. The situation consists of many segments of trajectories. The discontinuity of trajectories makes the particle filter framework unable to track or recall the subject after a long term disappearance. We calculate the association cost function from the state variables and texture signature, which is obtained by the tracking framework from Chapter 5. The detected subject can be either a new subject or re-appearing subject. The system has to categorise detected subjects based on cost measurements. Development and evaluation of the data association is discussed. From the experiment in Chapter 6, the recall rate is around 80% when tested by standard dataset and our own dataset.

Chapter 2

Background Theory

A tracking algorithm estimates the position of a dynamic object through two steps, prediction and verification. We consider the process of prediction and verification in this chapter and we try to classify various methods of tracking in order highlight the key ideas of the thesis.

2.1 Tracking

Tracking is an estimation process, which is normally considers the position of a desired subject. In order to estimate the position we need to perform two activities; prediction and evaluation. Evaluation can be made by calculating the distance or similarity measurements. Given some observation the system can verify a prediction by measuring the similarity between it and an observation. Detection is the simplest form of subject localisation. Detection has to test all possible positions in a considered space. Detection has to consider everywhere in the search space.

2.1.1 Similarity and distance

From the history of mathematics, several distance and similarity measurements have been introduced to tackle specific problems. Distance and similarity measurements are calculated normally in observation space, where the observation data is compared with the hypothesis data. There are many ways to measure the distance and similarity. In

order to choose a suitable metric we must know the data type of the observation, for example, whether the observation is expressed by a vector or a set, and the geometry of the data structure. When we deal with estimation such as curve fitting, where all samples are equally important, the squared Euclidean distance is normally selected as a error metric. The squared Euclidean distance always returns the positive and produces quadratic cost function, which is easy to be integrated with error optimiser, such as minimum mean square estimator.

A distance (or dissimilarity) of two vectors can be expressed by the distance. Let $dis(X, Y)$ denote the distance between vectors X and Y , which are vectors of real numbers in n dimensions $X \in \mathbb{R}^n$ and $Y \in \mathbb{R}^n$. The distance between two sets also can be computed. The two sets are denoted by A and set B , so the distance is $dis(A, B)$. We can determine the distance by choosing one from many methods *e.g.* [11–17], please see Table D.1. The table shows just a fraction of all possible formulas to compute the dissimilarity. Euclidian distance has been used extensively in geometry and square Euclidian distance has also been used in many regression methods such as linear regression. Cosine similarity is computed from the inner product and normalised by the magnitudes. The correlation similarity [13] is almost identical to cosine similarity. Sometimes counting the mismatched elements between two sets can form a useful measurement such as the Hamming distance [15]. Bitwise comparison between two sets is surprisingly important when dealing with missing elements of data or scaled images. The distance between two unequal cardinality sets can be measured by the Jaccard [16] or Hausdorff distance [17]. The selected similarity measurement or distance is varied depending on the assumption and application.

The similarity measurements or distances can reflect how well the prediction fits to the observation. And the ultimate goal of every kind of estimation is to measure the satisfaction between prediction and observation.

2.1.2 Prediction

In tracking problems [18], a *state vector* expresses position, velocity or the status of an object. To generate a prediction of the position of a target, all parameters involved in position transformation in the time domain must be added into the state vector. So,

prediction of the object state can be estimated by the state vector and the dynamic model of the object.

The simplest assumption is that a prediction of position in a successive frame is near to the previous position. This assumption is effective when object density is low. As long as the subject appears near by the prediction position, the tracker can follow the subject. However, a problem arises when the subject moves faster or within to a noisy environment. In a noisy environment, false detection can easily occur and the tracker is distracted by nearby objects. Once the tracker is distracted by other objects, it very likely to fail in the near future. The problem can be solved by including more information in the state vector. In a hyper-dimensional state space, the probability of two subjects being in the exactly same state is very rare. Therefore, increasing the dimension of the state vector reduces the distraction rate [19, 20]. Adding more information needs more computational power. So we need to find a good state representation to minimise the complexity.

A good prediction can reduce the searching space and increase the computational speed. When the complexity level of the state is limited by computational power of the hardware, increasing the dimension of state space is not a good solution. The state prediction from the prior knowledge is a fundamental key of modern tracking methods. The prediction defines a searching scope. In early research on multi-target tracking such as *joint probabilistic data association*[21] (JPDA) and *probability multi-hypothesis tracking*[22] (PMTH) , the area of high probability in which the subject could occur, also called *the gate*, was considered in order to prevent distraction by other objects. The gating technique is a data association method that allows a decision module to link any nearby subjects with in the scope of the tracker centroid. Basically, the gates removes irrelevance detections from consideration. The gate prediction allows us to narrow the search space and makes tracking more effective. A good model of prediction leads to high performance of tracking and detection. Further discussion on state prediction will be added in Section 2.2.2.

2.2 Review on Visual Tracking

In this section, we consider the ideas of prediction and verification. Verification needs a mathematical model or *appearance descriptor* (AD) in order to calculate the similarity. The AD is mathematical model or data structure to express the perception of the subject, which is obtained from observation images. The AD expresses detail of the observation and transfers to next process. The AD is a kind of language to represent the object. For example, “*a small red circle*” can express an image of an apple.

The prediction is an attempt to estimate unknown parameters. In visual tracking, the unknown parameters can be expressed by a *state vector*. The state vector expresses position and dynamic condition of the subject, which is usually independent from AD. For example, “*a small red circle on top left of the image*”. Hence we know both the AD (a small red circle) and the state vector (top left of the screen). If we have to track the apple on the screen, we know the kind of the object, and we have to find the unknown position.

In people tracking, the AD is usually independent from the state vector. In order to obtain prediction and verification functionalities, a tracking system consists of the AD and the state vector. We will separate the physical state vector from the AD and classify related work by these two features. Figure 2.1 shows two main ingredients (AD and the state estimator) of a visual tracking system. Visual tracking can be made from a AD and a estimator. We will discuss the combination of the AD and the estimation to make a tracking system in Section 2.2.1 and Section 2.2.2.

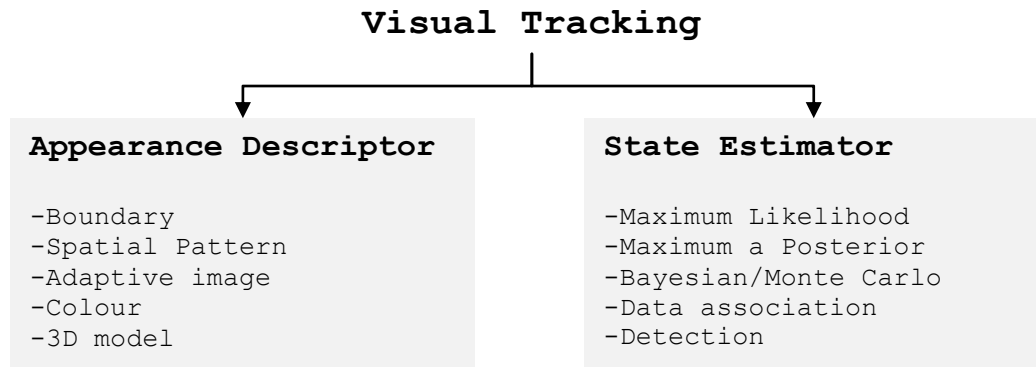


FIGURE 2.1: Tracking system consists of the appearance descriptor and the state estimator.

In this review, previous research related to visual tracking will be classified based on their ADs and state estimators. The AD is a composition of different data structures to express the appearance of a subject. A subject can be represented by a point, a region, a set of salient points, an image template, a colour histogram or a 3D model. The most complex descriptor expresses an accurate image of the subject but it requires huge computational power. There is a trade-off between accuracy and computational speed.

A projected image of a subject is constructed by two parts, the AD and the state. To minimise complexity of the system, the AD is normally assumed to be invariant in the time domain (and from different views in case of multi-view tracking). This assumption reduces the number of unknown parameters of the AD from the equation. The invariant descriptor is useful in solid object tracking, where shapes and corners are fixed with the centroid [23, 24]. However, in people tracking scenarios such an invariant descriptor does not exist or is impossible to express by a numerical model because a human body requires a huge number of parameters in order to express the constantly changing body and variation of clothes. As an invariant descriptor assumption cannot express reality. A possible solution is to learn the AD from the input sequence during tracking [25, 26] and express the subject by a collection of images. A similar approach was applied in multi-target tracking in [27].

The AD can be constructed in several ways to express perception of reality. The simplest form of the AD is a point or a coordinate in state space, all subjects are identical. We are possibly able to distinguish those subjects by their trajectories and current state but it is very ambiguous in practical situations. The method has been used in airborne radar tracking for decades [22] because radar instruments produced only the detected position. The ordinary method to classify airborne targets was to place a transmitter in the target, which sends a decoded signal to identify itself; it is also known as *secondary radar*. In *primary radar*, the airborne has no transmitter. In order to classify the object depth information is extracted by a synthetic aperture technique [28]. Because the point representation is indistinguishable, more appearance is required.

In static scenarios, state estimation can be done without prior knowledge from the previous state because the state is never changed. The simplest method to estimate the static state is using *maximum likelihood*. The maximum likelihood involves optimization

of state values that make maximum probability. In order to use maximum likelihood method the likelihood function must be assumed. If the distribution of the state values are Gaussian, the mean value that makes maximum likelihood can be computed from normal average [29].

In dynamic scenarios, the state of the subject is constantly changed. The dynamic prediction is required in this case. The Bayesian state estimator is an estimation method, which is able to include dynamic model of the state variables. The famous Bayesian state estimator, *the Kalman filter* [30], was introduced in 1960. The state transition in the time domain and the measurement were assumed to have additive Gaussian noise. The Kalman filter procedure consists of prediction and updating. Unlike maximum-likelihood, instead of using only current measurements to update the state, the update procedure involves the previous state in the time domain and current measurements. The current measurement is modeled to have a linear relationship with the current state. The current state depends on only the previous state, this model is also known as *a first order Markov process*. We will discuss about the estimation method in more detail in Section 2.2.2.

In next section we will explore the AD in visual tracking from related work. We classify pre-existing works by their appearance descriptor and state estimator as shown in Figure 2.2. Hence, we may discuss the same tracking method twice; the first time in the appearance descriptor (Section 2.2.1) and again in the state estimator (Section 2.2.2).

2.2.1 Appearance Descriptors

A gray-scale image expresses intensity of light as an array of pixels. Each pixel has an intensity value, which is normally stored in digital format. The raw image data is large and difficult to process directly. So we need a good descriptor to express objects. Appearance descriptors in the literature can be classified into five categories.

- Boundary describes a subject by lines and curves of its boundary
- Spatial pattern considers the spatial pattern in the subject region
- Adaptive image expresses a dynamic subject by a collection of images
- Colour considers spectrum property of the subject

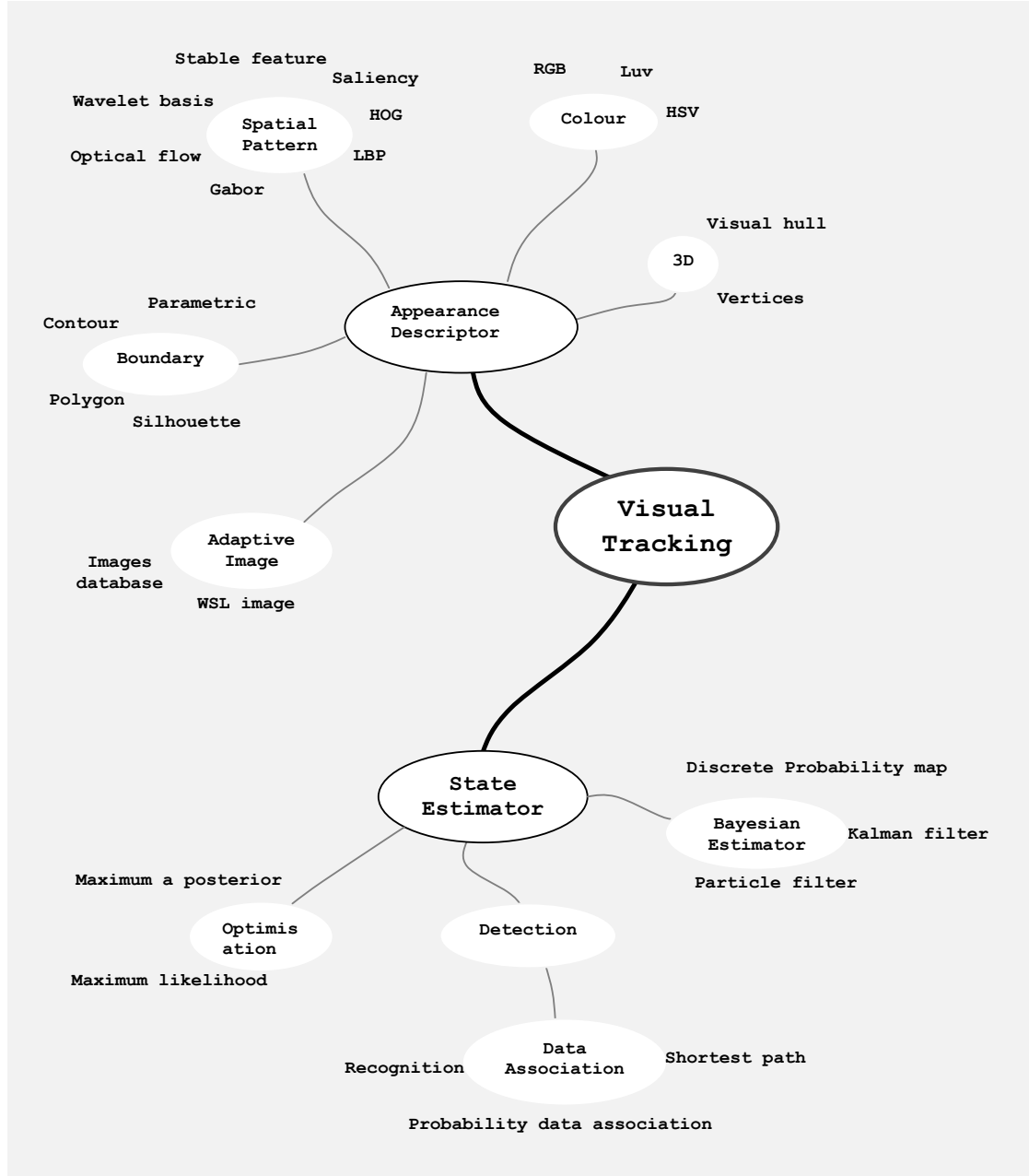


FIGURE 2.2: Components of visual tracking.

- 3D model includes 3D shaped, scale and orientation into an appearance descriptor

2.2.1.1 Boundary and Region

Canny edge detection [31] can extract edges from a gray-scale image. The light-weight appearance descriptor of the edge image is easy to be manipulated by a subsequent procedure such as an *active contour* method [6]. The active contour can also find out a subject boundary based on an intensity image.

Contour A contour represents the boundary of a subject. The subject image is transformed to an image of cost function, sometimes called *an energy*, that represents the boundary of the subject.

The active contour was introduced in computer vision in 1988 with the name *Snake active contour* [6] by Michael Kass and his colleagues. The contour was represented by points and lines between them. The Snake procedure is based on energy minimisation. The energy was determined from a linear combination between bending of a contour and intensity of the image at contour lines. By adjusting arbitrary coefficients in the energy function Kass could make the contour behave like a membrane or thin plate and it moved slowly converging to the edge or black lines on the image. The Snake active contour can be applying to visual tracking such as lips motion. However, the Snake active contour is unable to handle a large distance change of a subject relative to the previous position. The large displacement traps the contour points in a local minimum state. The Snake active contour is not suitable for tracking but it can play an important role in medical image segmentation, because the displacement of the organs does not change too much. Active contour idea were extended in many versions, *e.g.* [7, 32] In [7], Zezhi Chen extended the Snake idea by including colour dissimilarity between a subject template and a region in an image. The point-like contour in the Snake Contour was replaced by a binary bitmap image (all pixels are 0 except the pixels on contour are 1). Thus, the method can perform segmentation of an object in colour images efficiently. However, the large displacement problem is still disregarded. In [7], we can see that the centroid of a subject in image sequences is in almost exactly the same position.

Polygon The simplest way to express a subject is drawing lines around the subject. In [33], Kim use rectangular template to represent subjects with a particle filter. Tang [34] showed a tracking system using thermal imaging and a particle filter.

Parametric shape A circle and an ellipse can also be used to represent a subject. Parametric object detection such as *Hough detection* [35] can detect parametric shapes by processing an intensity image or an edge image. An ellipse is a flexible shape (height, width and orientation) that can fit many forms of the subjects. Because of flexibility and a small number of parameters, the ellipse model was used in tracking in many works [36–39].

Region and Silhouette Unlike the boundary representation, the region representation considers the pixels inside the boundary. This representation is closely related to the segmentation method, a process to classify individual pixels into subsets based on information at pixel-level. The segmentation method extracts objects from an image of a complex scene. Segmentation can use various kinds of information to compute a silhouette of the subject, for example, an intensity image [40], texture information [41], a depth image [42], colour in Luv space [43], *etc.*

2.2.1.2 Spatial pattern

The spatial pattern is computed by convolution[44] between an image and a 2D pattern template. The method to extract the spatial pattern is also called *a convolution filter*. The convolution filter also have different name depending on the pattern template, such as the gradient filter, the corner filter, the local binary patter filter [45], *etc.*

Optical flow is a tracking process at pixel-level. There are two classical methods to compute the optical flow, the Horn-Schunck[46] and Lucas-Kanade[47] methods. They are based on constant illumination and velocity smoothness. Horn and Schunck solved the velocity vector or flow by the calculus of variations. In contrast, Lucas and Kanade used the least square method. It is inconvenient to call these two methods “tracking” because they don’t actually follow a object. In fact, the methods compute flow for every pixel. In order to perform tracking, optical flow has to be combine with a segmentation method. Cooperation between optical flow and segmentation can be found in [48, 49] (optical flow with active contour), [50] (optical flow with background subtraction), *etc.*

Saliency can be detected by a feature detector such as a corner detector[51] or a Laplacian filter. Unlike optical flow the feature point representation considers only high detail surfaces, also known as *saliency surfaces*. Some feature point tracking methods were extended from the optical flow methodologies such as [52, 53]. The appearance descriptor in this class consider only the high detailed area of a image.

Stable feature points Normal feature detection is sensitive to change in scale and orientation. In order to perform tracking or recognition we need a stable feature, which

is invariant to scale and orientation. The scale-invariant feature transform, or SIFT [54], is a scale and orientation invariant feature detector. SIFT used Difference of Gaussian (DoG) filters to keep the desired frequency feature points from many feature points in a image, called *extrema in scale space*. Then it detects saliency by the Laplace filter. Because DoG is highly sensitivity to edge and appearances of edges are changed by rotation, all edges need to be removed. The remaining feature points is expressed by a magnitude of gradient and the orientation. The SIFT is stable in scale and orientation transformation so it can recognise static object robustly [54, 55]. Herbert Bay improved the speed of SIFT by using Haar wavelets and called the new version SURF (Speeded Up Robust Feature) [56]. SURF is significantly faster than SIFT and it has been used in many applications, for example in tracking [57].

Wavelet basis The wavelet feature or basis is a spatial frequency orthogonal basis, where each feature is orthogonal to each other (convolution of two different bases equals zero). Viola and Jones developed fast Haar-like basis computation by using *the integral image* [58, 59]. The basis feature and the integral image makes detection and tracking faster [26, 60].

HOG The Histogram of Oriented Gradient (HOG) [61] is a vector feature, where each element of the vector is a bin of a histogram constructed by counting a gradient vector in particular direction in an image region. The HOG was applied to people detection in [61] and the result was reliable with detection rate 84-89% at 0.01% false alarm.

LBP The Local Binary Pattern (LBP) is a binary coding to describe the relative intensity of neighbour pixels. It was developed for texture classification [45, 62]. It was extended for people detection and called *semantic LBP* [63].

HOG+LBP Combining HOG and LBP was studied to improve people detection. This study showed reliable results with 97.9% detection rate at 0.01% false alarm. People detection is applied to people counting in [64].

Gabor feature A Gabor function is an oriented wavelet pattern. The Gabor filter has been found in the primary visual cortex [65, 66]. Daugman [67] stated that “Simple

cells in the visual cortex of mammalian brains can be modeled by Gabor functions”. Gabor features were applied to face recognition and gave reliable results [68, 69]

2.2.1.3 Adaptive image

Database of images Collective appearance descriptors can be acquired during the tracking process. Since, the subject appearance is dynamic, a static template cannot express the reality of a constantly changing shape of a human body. A pool of templates could be cumulatively acquired during the tracking process. The data structure of a template pool is a key to make fast similarity computation. In [25, 70], an average and an Eigenbasis of images were used as templates and these templates were generated during the tracking process. Using different sets of averaged images is a fast and effective solution. However, computing the Eigenbasis of an image is a time-consuming task.

Constructing the template pool during tracking also appeared in in Kalal’s works [26, 60]. Kalal used a Binary pattern (a Haar-like feature) of a gradient map as a appearance descriptor. The descriptors are generated and saved to long-term memory by detection, tracking and learning mechanisms. In the tracking module an observation image in the tracker boundary is compared to all features in the database. To perform matching between the dynamic appearance and the template in the database, Kalal proposed three strategies.

- *Absolute*, the system accepts a new feature when the similarity between the new feature and the stored features is larger than a threshold. In other words, the a distance between two appearances in feature space is smaller than a threshold.
- *Change*, the association module checks continuity in feature space between a previously accepted feature’s volume and the current observation. It makes the acceptance region dynamically grow larger during the tracking process.
- *Loop*, when the acceptance volume in feature space spreads and eventually converges to the initial location it is called *a loop*. In my opinion, the Loop accepting strategy is redundant because the Change strategy can describe the Loop. So the Absolute and Change strategies are capable to the classify whether to accept or reject the observation.

We can imagine that the acceptance feature volume keeps growing. Eventually it could increase the false alarm rate because the volume is too large. In my opinion, the acceptance volume must be regulated by some mechanism to prevent over growing of the acceptance volume in the case of tracking over long period tracking.

WSL image The WSL (Wonder Stable Lost) model is a probabilistic model to express appearances of a subject occurring in a tracking sequence. The WSL model was firstly introduced by Jepson in 2003[71]. The WSL consists of three modes *stable*, *lost* and *wandering*. The *stable mode* images normally had small changes so the magnitude of the wavelet feature [72] in the considered region is stable and the appearance can be described by a Gaussian distribution. During tracking, the feature could be occluded and the event was modeled by *the lost model*. The constantly change features did not contribute to the tracking process and it was modeled as *the wandering model*. The WSL model was used in other works, *e.g.* [73, 74].

2.2.1.4 Colour

Colour information is distinctive and can represent physical properties of surfaces. The colour histogram was used in the mean-shift tracker[75], which consider colour histogram of pixels inside a region. The mean-shift is widely used because of it's simplicity and robustness. The mean-shift will be described in detail in Section 2.3.

Colour is a useful information and there are so many ways to interpret colour. Colour distribution on spatial domain is a useful combination to track and identify people. For example, a cylindrical model with colour histograms for each vertical section of the cylinder was a human model in [27]. Coloured rectangles were used in an articulated human model [76]. Colour is a good method to separate sporting players from the field [77], by using a colour segmentation method. Colour distribution on an image, which is called *spatiogram*, were use for object tracking in [78]. Extension of the spatiogram tracking was studied in [79].

2.2.1.5 3D model

Vertices A human shape can be represented by a 3D model. In computer graphics the 3D model is usually modeled by a set of vertices (a vertex is a point that links to other points to form a polygon plan). The vertices are placed on an object surface in 3D space. The vertices model was used in a solid object tracking application [24]. In fact it can be used in any kind of object tracking, such as faces [80], people [81], *etc.*

Visual hull A 3D subject's surface can also be represented by many silhouettes from different points of view. It is also called *a visual hull*. This method can be used in order to estimate positions of subjects. For example, Berclaz [82] computed *the probability occupancy map* (POM) by using silhouettes from different views and the POM was used in the subsequent Bayesian tracking method. The silhouette can be computed by a background segmentation method [83, 84]. POM was also used in other work, *e.g.* [85].

2.2.2 State Estimators

State estimation related to visual tracking can be divided into five classes.

- *Maximum Likelihood* (ML) has no probability and transition model. The current state depends only on the current measurement.
- *Maximum a Posteriori* (MAP) obtains the state estimation by Bayesian statistics and the resulting estimation is expressed by a single state that maximises the posterior probability.
- *Bayesian Probability* estimates the state by Bayesian statistic similar to MAP but the result is represented by a posterior density instead of a single point of state. The benefit of this method is that it can tell possible confidence region in the state space, which is a single point cannot.
- *Data Association* has been introduced as early as 1995 [22]. The data association method creates trajectories of the subjects by linking detected locations of the same subject together. In [22], the nearby detections to a previous trajectory were considered as the candidates and a probability model was modeled for selecting the best candidate. The method has been improved by adding a complex appearance

descriptor [26, 86]. Normally the data association method requires detection as a primary input.

- *Detection* has a large scope search space and involves template similarity and classification. The template similarity can be anything from a simple threshold classification to a sophisticated discriminative learning model [61, 63, 87].

2.2.2.1 Maximum-Likelihood

Some tracking methods [6, 7, 32, 42, 75, 78, 79, 88] rely on the simplest assumption that the position of an object in the future is near to the position of the current state. Hence, the estimation of position depends on only the current observation regardless to the previous state.

$$S_{ml} = \underset{S}{\operatorname{argmax}} \{ \mathcal{L}(X|S) \} \quad (2.1)$$

Equation (2.1) shows maximum-likelihood notation, where the likelihood function $\mathcal{L}(X|S)$ is computed from a dataset X and the state S . The likelihood function need to be presumed by probabilistic modeling and the appearance descriptor. An object or a person can be represented in many ways, so the likelihood (which is defined by the appearance descriptor) also can be expressed in many ways. In order to estimate the state, ML computes the optimum state (S_{ml}) that makes the synthetic appearance best match to the observation image without using a prior probability function.

A person's appearance can be expressed by a profile of their shape, colour histogram, texture and etc. With the appearance descriptor of an observation, we can make an algorithm perform tracking by maximising similarity. In maximum likelihood, the motion model is excluded. Including a motion model in the algorithm could not improve the result. The optimizer will search the local optimum and ignore the prediction from the motion model.

Tracking by maximising likelihood will return a local optimum position of the highest matching score between the observed AD in the tracker boundary and the template. The boundary expresses the position and shape of the subject. The tracking algorithms in this class perform as local optimisers, which search for a local optimum state. This means the tracking method is likely to be trapped in a wrong local optimum when the

landscape of the similarity function is changed quickly due to fast subject motion or occlusion.

Examples of this class are *mean-shift tracking* [75], *spatiogram tracking* [78, 79] and *active contour* methods [6, 7, 32, 37, 42]. These methods have no motion model and no predicted position from the previous frame. Lack of prior position can make these methods suffer from failure due to large displacement, fast motion or low frame rate. The tracking methods in this class make one strong assumption that the appearance from the previous frame is similar to the next frame. The assumption is too strong and does not allow the subject to be absent, leading to failure when occlusion happens.

2.2.2.2 Maximum a Posterior

The maximum a posterior (MAP) applies the prior probability function in estimation and the estimated state is represented by a single state vector. The method is very similar to the maximum likelihood but the use of prior $\mathcal{P}(S)$ is included in the estimation as in Equation (2.2).

$$S_{map} = \underset{S}{\operatorname{argmax}} \{ \mathcal{L}(X|S) \mathcal{P}(S) \} \quad (2.2)$$

For example, *the expectation maximization* (EM) algorithm consists of two steps, an estimation step and a maximisation step. In the estimation step, an expectation of the state is made (it is considered as prediction) and the maximum-likelihood is done in a maximisation step. The EM algorithm was proposed by Dempster in 1977 [89] to tackle the incomplete data problem in statistics. The incomplete data could be for example the number of clusters or mean values of the considered population. This incomplete problem is very similar to our tracking problem. The EM algorithm was also applied to tracking applications, *e.g.* [71].

The similar technique, which involves prediction and likelihood-maximisation, can be estimated by Bayesian interference. Some classic trackers, *e.g.* [22], applied Bayesian interference in the estimation. However, the output of the estimation is a single state and the estimation was done by cooperation between prediction and the maximum-likelihood method. So, we can classified as a maximum a posterior (MAP) method. The MAP

estimation represents the estimated state by a single vector, which makes calculation fast, *e.g.* [40, 90].

2.2.2.3 Bayesian Estimator

In Bayesian estimation the posterior keeps the original probability density representation. The posterior density is proportional to a product between the likelihood and prior density as in Equation (2.3). This estimator involves probability integration and sampling method as will be discussed in more detail in Section 2.5.

$$\mathcal{Q}(S|X) \propto \mathcal{L}(X|S) \mathcal{P}(S) \quad (2.3)$$

There are many ways to implement the Bayesian estimator, *e.g.* Kalman filter[36], Particle filter [23–25, 33, 34, 70, 73, 80, 81, 85, 91] or representing density by grid-space-map as in [27]. We will discuss Bayesian estimator again in more detail in Section 2.5.

2.2.2.4 Data Association

Detection and association are also solutions to the tracking problem. The strategy of this method is similar to the name, which involves detecting subjects and associating present detections with previous trajectories.

Probability data association Joint probability data association (JPDA)[21] and *Probabilistic Multi-hypothesis Tracking* PMHT[22] are fundamental to modern multiple-target tracking. They were designed for radar tracking. The radar instruments at that period provided only detected positions. The idea of probability data association is based on probability propagation in the state space. The associations were computed to maximise the joint posterior probability of pairing between every individual trajectory with every given detected position. In 1995, Roy Streit [22] proposed the PMHT. The PMHT generated many hypotheses for a subject’s motion. To associate an observation to a subject’s trajectory, PMHT used a gate, a circle in state space, which it could expand due to uncertainty of estimation or covariance from a Kalman filter.

Since people detection, *e.g.* [61], has been developed, the cooperation between object detection and probability data association has been widely used in people tracking applications, *e.g.* [86, 92]. The process starts with the detection module processing an input image and passing all detected positions to a data association module. The detected locations are matched to the previous trajectories by the association module by optimising the probability or the cost function. The cost function can be modeled as a probability diffusion in the state space [77, 82, 86]. To achieve a high accuracy the data association tracking method relies on the performance of detection.

Detection modules reduce a half million pixel image to a few detection points. Each detection point is classified as a false-alarm or a correct detection and then it is joined to the existing trajectory or created as a new trace. Bayesian estimator such as Particle filters or MCMC were used for optimising the association probability as in [77, 86].

In [82], a new detection is matched to a previous position by using the k-shortest path. A temporal spatial grid of detection was saved and then a graph was generated from the grid. A probabilistic model was used for computing the cost function between nodes. A disjoint path algorithm was applied to find minimum cost paths. This association involved probabilistic modeling and graph optimisation.

Shortest path In order to reduce complexity, the subject is represented by a single vector and ignores the density function. A single point representation is light weight and deterministic. Single target tracking can be done by a shortest path algorithm. In [93], Yan showed how the Dijkstras shortest path algorithm was applied to data association for tracking a tennis ball.

Multiple target tracking can also be computed by the data association method. When the detection position is extracted from a raw image we can associate newly detected points with existing trajectories. The simplest approach is to join a detected point to the closest trajectory, similar to [93]. The computational complexity of combinatorial calculation in data association increases with the number of trajectories and subjects. For example, if we have N detected points and N trajectories the combinatorial association can be computed in polynomial time $O(N^3)$ [94]. The problem is more difficult when the distribution of detection points is dense and the number of trajectories is unequal to the number of detected points. The association module must assign each

new detected point to an individual trajectory and separate the false-alarms out from consideration. Each connection has a cost, which is defined by the distance between the previous trajectory and the detected point.

Matching N subjects to M categories is known as the *assignment problem* in logistics. In 1955, Harold Kuhn [95] suggested that the Hungarian algorithm can be applied to solve the assignment problem. However, the Hungarian algorithm described by Kuhn was written in a manuscript which was implicit and not designed for computer programming. In 1957, James Munkres [96] polished the Hungarian method and made a clear explanation for any programmer.

A disadvantage of tracking by detection and data association is that a single point cannot represent a probability density function. When a subject is visually blocked or distracted, the observation is unclear and it is difficult to make a decision from the current frame. The decision needs further observation. Some extra observations will be measured in the next consecutive frames in order to gain confident to make a decision. It is better to keep using the probability density and to not make any decision at any uncertain step, which has insufficient observation for creating the subject trajectory. Therefore, any method that applies best assignment can fail in a difficult situation and an error will be amplified in the next frame [97].

Recognition Association between a previous trajectory and a detected subject can be achieved by recognition. Single target tracking by recognition data association can be found in [26, 98] However, there is lack of study in recognition in a multi-target tracking scenario. In Chapter 6, we consider data association by the recognition in multiple-target tracking context.

2.2.2.5 Detection

Background Segmentation The simplest method to detect a moving object is to separate them from a static background by exploiting the technique called *background segmentation*. An adaptive mixture of Gaussian (MoG) background subtraction [83] is light weight and adaptable to changing illumination conditions. It automatically learns and re-initialises the background model of intensity for every pixel. For pixels of the static background, the intensity is described by a Gaussian distribution with a mean and

a variance. The background model of a colour video sequence can be done with the same technique. Once the background model is obtained, the moving object can be extracted. The extraction method is called *segmentation*. Segmentation can be used in order to detect moving objects. Other benefits of the MoG background segmentation method are low complexity and pixel independent computation, which are easy to implement with parallel programming.

The Wallflower [84] was another background segmentation method designed for high accurate segmentation. The three layers of computation (pixel, region and frame) make Wallflower highly complex. The dependency of the algorithm makes it difficult to implement on a parallel platform. Background subtraction is a cheap and useful way to detect a moving object. However, the cameras must be fixed in position. The static camera constraint makes the method unpopular.

People detection The *histograms of oriented gradients* (HOG) human detection was introduced by [61]. An improvement [99] of HOG offered high speed processing and ability to detect multi-scaled people in an image. HOG detection [100] was implemented on a GPU GeForce GTX 285 to process one 640×840 image at a maximum speed of 74ms (13fps). Improvement of people detection methods can be found in combination of many features. For example HOG and LBP people detection in [64, 87] or visual hull method in [101].

2.3 Mean-Shift method

The mean-shift method was introduced as a peak finder and later on it was applied to the visual tracking problem. Mean-shift is a non-parametric iterative method to find a peak of a probability density [102, 103]. The kernel is a standard template of a probability function. The kernel $K(x)$ can be any standard density function such as uniform, parabolic or Gaussian.

The mean-shift has been used for seeking the peak of a density function $P(x)$. The illustration in Figure 2.3 shows how kernels move according to a density function from a previous expectation value μ_o to the new expectation value μ . The new expectation of x in a density function is computed from a product between $P(x)$ and a kernel at

previous location μ_o . The mechanism performs peak seeking similar to gradient ascent, where the kernel moves towards the high density area. From the Cheng study Equation (8) in [103], the update equation of the mean in integral form is Equation (2.4).

$$\mu = \frac{\int x.P(x).K(x - \mu_o)dx}{\int K(x - \mu_o)dx} \quad (2.4)$$

The mean-shift is useful when dealing with a distributed dataset and we have to estimate the peak of $P(x)$, where the data population distributes as an unknown density $P(x)$.

In short, the mean-shift computes the new mean by using the previous mean and the integration over the spatial space. In the case of a single peak density function, the new mean always moves to a higher density area. If the density function at position μ_o is increasing respective to x , the kernel will move to the right and vice versa. Therefore, if we keep updating the mean in this fashion the new mean will stop at a peak of the density function. The property allows us to perform gradient ascent to search for a peak as in Figure 2.3.

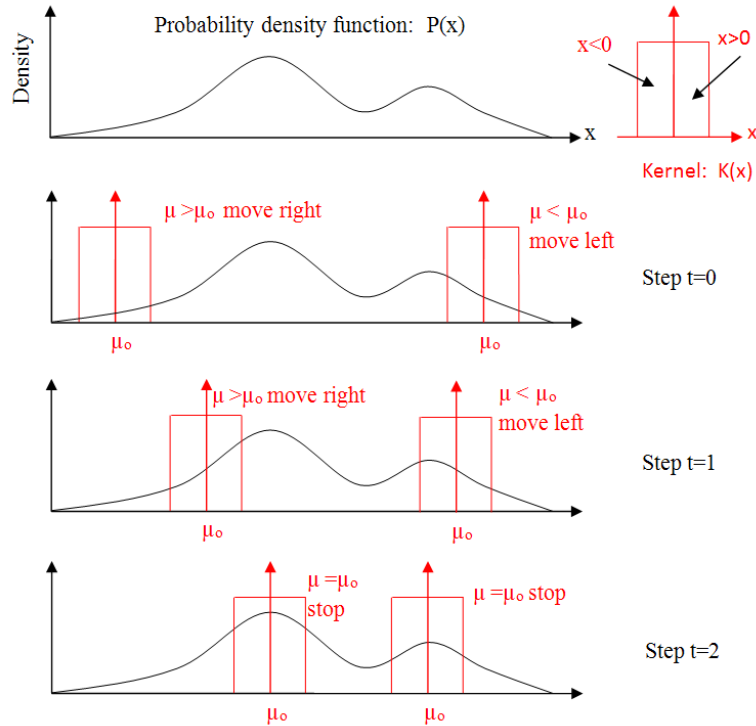


FIGURE 2.3: Mean-shift seeking density peaks. In this example, uniform kernels are applied to perform gradient ascending over density function resulting in two stationary points at final step.

In 2000, Dorin Comaniciu [75] applied the mean-shift to visual tracking . In his work, the density function $P(x)$ is replaced by a similarity function called The Bhattacharyya coefficient. The Bhattacharyya coefficient is the similarity between a template colour histogram, $\mathbf{m} = [m_b]; b = 1, 2, \dots$, and a current colour histogram of the tracker region, $\mathbf{n}(\mu) = [n_b(\mu)]; b = 1, 2, \dots$, where the tracker at position μ is computed from Equation (2.5), where b is a histogram bin index.

$$P(\mu) = \sum_b \sqrt{m_b \cdot n_b(\mu)} \quad (2.5)$$

The histogram \mathbf{m} and $\mathbf{n}()$ is weighted by an Epanechnikov kernel. A function δ_{ib} is 1 when colour vector at position x_i falls in a histogram bin b , otherwise 0. So a histogram of colour can be computed by summation of δ_{ib} for all i . A bin b of the centre weighted histogram $\mathbf{n}(\mu)$ was defined as Equation (2.6), where $K()$ is an Epanechnikov kernel.

$$n_b = \sum_i K\left(\frac{x_i - \mu}{h}\right) \delta_{ib} \quad (2.6)$$

$$K(x) = \begin{cases} \frac{3}{4}(1 - x^2), & |x| \leq 1 \\ 0, & |x| > 1 \end{cases} \quad (2.7)$$

The vector \mathbf{n} is a vector representing colour histograms of pixels in the tracker kernel region. If we maximise $P(\mu)$ by changing μ , we will get strong similarity between vector \mathbf{m} and \mathbf{n} . In order to maximise similarity, Comaniciu used a Taylor's series to expand the Bhattacharyya coefficient when \mathbf{n} is slightly changed.

$$P(n_b(\mu)) \approx P(n_b(\mu_o)) + \sum_b \{n_b(\mu) - n_b(\mu_o)\} \frac{\partial P(n_b(\mu_o))}{\partial P(n_b)} + \dots \quad (2.8)$$

$$= P(\mu_o) + \sum_b \{n_b(\mu) - n_b(\mu_o)\} \frac{\partial(\mu_o)}{\partial n_b} \quad (2.9)$$

$$= \sum_b \sqrt{m_b n_b(\mu_o)} + \frac{1}{2} \sum_b \{n_b(\mu) - n_b(\mu_o)\} \sqrt{\frac{m_b}{n_b(\mu_o)}} \quad (2.10)$$

$$P(\mu) = \frac{1}{2} \sum_b \sqrt{\mu_b n_b(\mu_o)} + \frac{1}{2} \sum_b n_b(\mu) \sqrt{\frac{m_b}{n_b(\mu_o)}} \quad (2.11)$$

The derivative of P is set to zero.

$$\frac{d}{d\mu}P(\mu) = \frac{1}{2} \sum_b \sqrt{\frac{m_b}{n_b(\mu_o)}} \cdot \frac{d}{d\mu} n_b(\mu) \quad (2.12)$$

$$0 = \frac{3}{4} \sum_b \left[\sqrt{\frac{m_b}{n_b(\mu_o)}} \sum_i \frac{x_i - \mu}{h^2} \delta_{ib} \right] \quad (2.13)$$

$$0 = \sum_i \left[(x_i - \mu) \sum_n \delta_{ib} \sqrt{\frac{m_b}{n_b(\mu_o)}} \right] \quad (2.14)$$

$$0 = \sum_i (x_i - \mu) w_i \quad (2.15)$$

where

$$w_i = \sum_b \delta_{ib} \sqrt{\frac{m_b}{n_b(\mu_o)}} \quad (2.16)$$

From Equation (2.15), he computed a new mean. The optimal new mean can be written in short form as Equation (2.17).

$$\mu = \frac{\sum_i x_i w_i}{\sum_i w_i} \quad (2.17)$$

This approach is similar to Newton's method in optimisation: first, expand the cost function by a Taylor series then set the derivative of series to zero and solve for the new location.

Mean-shift tracking is well known because of it's low computational complexity and high accuracy. It is a combination between a similarity measure and optimization. However, mean-shift tracking is unable to solve some practical problems, for example, rotation around an axis which is perpendicular to camera direction. Rotation reveals a new surface of the subject and the colour histogram of the new surface is often different to the template. Distraction from other surfaces, which have similar colour, and occlusion can still lead to failure of this method. Occlusion, motion model and short-time disappearance were disregarded in this method.

2.4 Kalman Filter

The Kalman filter [30] is a well known Bayesian estimator, which has been applied in machine control intensively for over five decades. A Kalman filter is a recursive

error optimizer where distributions of the state variables are assumed to be Gaussian distribution. It was designed to estimate the state variables of a linear stochastic system (a linear system with some noise perturbation).

The Kalman filter was proposed by Rudolf Emil Kalman in 1960 [30]. The original Kalman filter was designed for linear system, where the state vector \mathbf{x} contains information of target subject such as position and velocity. A measurement vector \mathbf{y} contains information detected by sensors. The vector \mathbf{y} is modeled by state \mathbf{x} and a projection matrix \mathbf{G} . Usually the state of the system is unobservable, so, \mathbf{G} expresses a physical relation between the hidden state and the measurement vector.

$$\mathbf{x}_t = \mathbf{F}\mathbf{x}_{t-1} + \mathbf{v} \quad (2.18)$$

$$\mathbf{y}_t = \mathbf{G}\mathbf{x}_t + \mathbf{w} \quad (2.19)$$

The prediction of a state is generated from a previous known condition by the *transition function* \mathbf{F} . The prediction is also known as *the prior*, which needs to be verified by observation. The present state vector \mathbf{x}_t linearly depends on the previous state vector \mathbf{x}_{t-1} and a Gaussian noise perturbation \mathbf{v} . A linear operator \mathbf{F} transforms the previous state \mathbf{x}_0 to the new state \mathbf{x} as in Equation (2.18). The prediction could be as simple as ballistic motion. Measurement \mathbf{y} is a function of \mathbf{x} , where the observation function \mathbf{G} transforms state \mathbf{x} to measurement \mathbf{y} . The measurement also has additive Gaussian noise \mathbf{w} .

The Kalman Filter has prediction and update steps. The state vector x is a stochastic variable so it is expressed by mean μ and covariance Σ . In the prediction step, these Gaussian parameters (μ_t and Σ_t) are predicted by linear transformation \mathbf{F} . The covariance is transformed by Equation (2.21). \mathbf{Q} is the covariance of the noise, $\mathbf{Q} = \langle (\mathbf{v} - \bar{\mathbf{v}})(\mathbf{v} - \bar{\mathbf{v}})^T \rangle = \langle \mathbf{v}\mathbf{v}^T \rangle$. Note that the covariance \mathbf{Q} and \mathbf{R} can be fixed or varying depending on an individual application.

$$\tilde{\mu}_t = \mathbf{F}\mu_{t-1} \quad (2.20)$$

$$\tilde{\Sigma}_t = \mathbf{F}\Sigma_{t-1}\mathbf{F}^T + \mathbf{Q} \quad (2.21)$$

Then the predictions $\tilde{\mu}$ and $\tilde{\Sigma}$ are corrected by supporting observations as in Equation (2.22) and Equation (2.23). The predicted mean $\tilde{\mu}_t$ is corrected by error $(\mathbf{y}_t - \mathbf{G}\tilde{\mu}_t)$ with the *Kalman gain* \mathbf{K} . The covariance is also reduced by factor $\mathbf{K}\mathbf{G}$.

$$\mu_t = \tilde{\mu}_t + \mathbf{K}(\mathbf{y} - \mathbf{G}\tilde{\mu}_t) \quad (2.22)$$

$$\Sigma_t = (\mathbf{I} - \mathbf{K}\mathbf{G})\tilde{\Sigma}_t \quad (2.23)$$

The Kalman gain \mathbf{K} is computed from the equation below.

$$\mathbf{K} = \tilde{\Sigma}_t \mathbf{G}^T (\mathbf{G} \tilde{\Sigma}_t \mathbf{G}^T + \mathbf{R})^{-1} \quad (2.24)$$

$$\mathbf{R} = \langle \mathbf{w}\mathbf{w}^T \rangle \quad (2.25)$$

The learning rate of a Kalman filter is adaptive as it varies from time to time according to covariances Σ_t and \mathbf{R} . The Kalman gain matrix \mathbf{K} controls the learning rate of the Kalman filter. In [18], Welch and Bishop showed that when the covariance of observation noise \mathbf{R} is near zero, the Kalman gain \mathbf{K} will converge to the inverse of matrix \mathbf{G} .

$$\lim_{\mathbf{R} \rightarrow 0} \mathbf{K} = \mathbf{G}^{-1} \quad (2.26)$$

And if the covariance \mathbf{R} is large the gain will drop to zero and makes no update.

$$\lim_{\Sigma_e \rightarrow 0} \mathbf{K} = \mathbf{0}. \quad (2.27)$$

In practice, the covariance \mathbf{R} is initialised to be relatively large, so \mathbf{K} is about \mathbf{G}^{-1} . Eventually \mathbf{K} gradually approaches zero by reduction of the covariance \mathbf{R} .

Uncertainty increases in prediction and decreases by verification. The determinant of a covariance matrix expresses the uncertainty of the stochastic variables. In the prediction step of Equation (2.21) because of noise \mathbf{Q} , the determinant of the predicted covariance $\tilde{\Sigma}_t$ is increased from the previous state, $\det(\Sigma_{t-1}) < \det(\tilde{\Sigma}_t)$. In the correction step, Equation (2.23), the determinant of the predicted covariance is smaller or equal to the corrected covariance, $\det(\tilde{\Sigma}_t) \geq \det(\Sigma_t)$. One can prove this by using the fact that all covariance matrices are positive definite matrices and the determinant of a covariance matrix is a positive number. The determinant of a covariance matrix expresses the volume of uncertainty in state space. The uncertainty increases in the prediction step

and decreases in the update step. After, several iterations the uncertainty converges to an equilibrium that is characterised by the \mathbf{Q} , \mathbf{R} and the observations.

The Kalman filter was modified for non-linear problems because many practical problems in engineering are non-linear. The *Extended Kalman filter* approximates a transition function \mathbf{F} by the first-order Taylor series, which is useful for approximating non-linear systems. Another variation the *Unscented Kalman filter*[104] transforms the stochastic variables, in which makes the nonlinear becomes a linear problem.

The Kalman filter cannot perform tracking without an detection module. Normally the measurement vector is produced by an instrument. For example a radar system detects aircraft and sends the position to a Kalman filter to estimate the state vector. Similarly, an object detector can detect a object from a video sequence and pass the detected position to Kalman filter. Tracking by using the Kalman filter relies on the accuracy of the detector. We have to suppress the miss detection rate and false alarm rate to be significantly small. Applying a Kalman filter to visual detection rely on accuracy of the detection. So the detection module is a necessary part of Kalman tracking. A people detection module is normally constructed by matching between a template and an observation. The traditional Kalman tracking considers the state estimator and detection separately.

In addition, the Kalman filter does not allow us to exploit visual features and is unable to make full utilization of the images. The detection module cut out so much useful feature information. In multiple target tracking, distraction cannot be fully solved by using only information of detected position, especially, when two subjects are close and moving in the same direction. The state vectors of these subjects are very close in state space. So distraction is inevitable in position-only tracking. Therefore, applying a Kalman filter alone to multiple target tracking suffers from serious distraction. Other features must be used in order to identify the subjects. Without a feature vector it is difficult to follow the same subject amongst many subject, where all subjects are represent by points.

In literature, single target tracking can be done by cooperation between the Kalman filter and feature points detection, *e.g.*, [105, 106]. Kalman tracking is a very light weight computation and works quickly. However, using a Kalman filter in multiple target tracking is fairly problematic. A comparison in [107] showed that the performance of

a Multiple Hypothesis Kalman Filter is lower than a Hybrid Joint Separable Particle filter. Distraction from nearby objects confuses the system and it is unable to associate each observation to each tracker. The particle filter or a numerical Bayesian estimator will be discussed in Section 2.5.

2.5 Numerical Bayesian estimator

Bayesian tracking framework represents the state by a probability density. Unlike Kalman filter, the probability can be any shape it is not strict with Gaussian distribution. The posterior density (confirmation) is a product between prior density (prediction) and likelihood function (similarity measure). The prior density defines the boundary of search in state space. The confined boundary improves the searching efficiency compared to exhaustive search in normal detection method. Surely, exhaustive search will take a long time to complete all possible locations especially in a high dimensional state space. Prediction can reduce the search volume. But the problem is how we can generate prediction.

The state estimator needs a probabilistic model to predict a future state from the current state. The transition process of the state in the time domain is known as a stochastic process. Stochastic modeling is common when we are dealing with uncertainty and we can find it in many fields of study, *e.g.* in finance [108], biology [109] and engineering [110].

A common method to generate a prediction is combining deterministic information and randomness. The deterministic solution from a mathematical model gives an expectation value of the state, where the perturbation of an unknown factor is represented by random noise in state space. Given a particular density function of the previous state, the added noise makes the shape of the density function wider (the variance increases). However normally the added noise has zero mean, which has no effect on the mean of the density after addition. In a small interval of time the subject motion is similar to ballistic motion and we can assume that the new position depends on the previous position and velocity. Tracking by a probabilistic model usually uses prediction and similarity to generate a confirmed density function. In Bayesian statistical inference the distribution of prediction, similarity and confirmation is called as *the prior*, *likelihood* and *posterior*

densities, respectively. Bayesian theorem of conditional probability describe that the posterior density function increased linear proportion to a product between the prior and likelihood density functions.

In the maximum likelihood method, a large set of observations is stored and the standard average and variance will be computed according to the presumed likelihood distribution. A drawback of the maximum likelihood method is that it needs large memory storage. In a real-time estimator such as the Kalman filter, an observation is used once in an iteration and then it will be deleted. Only the state is stored in memory and this state will be used in the next iteration. This method is also call *recursive Bayesian estimation*.

To explain prior, likelihood and posterior let us consider an example. Imagine we are going to estimate the distance of a static object by using a sonar instrument which returns a distance measurement for every time frame. Due to wind turbulence, the instrument rarely gets a exact distance between the observer and the subject. So we estimate the distance by applying a probabilistic interference model. Let the observation of a position vector be denoted by x . Assume the wind turbulence makes the observations distributed as a Gaussian function with mean μ and standard deviation σ . An objective is to recursively estimate μ and σ from a series of observations. In this case, x is a measurement and state of the system are μ and σ . The stochastic state variable of x is represented by a Gaussian function with a mean μ and a standard deviation σ . The prior density $\mathcal{P}(\mu, \sigma)$ is a function of μ and σ . The likelihood function is defined as the probability of measurement set x to occur given prior parameters of the probability density function [111]. So the likelihood is simply a probability on the Gaussian distribution function. The likelihood function is denoted by $\mathcal{L}(x|\mu, \sigma)$.

$$\mathcal{Q}(\mu, \sigma|x) = \frac{\mathcal{L}(x|\mu, \sigma) \mathcal{P}(\mu, \sigma)}{\mathcal{P}(x)} \quad (2.28)$$

$$\mathcal{L}(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \quad (2.29)$$

$\mathcal{P}(x)$ is difficult to determine because it depends on a particular problem, which is difficult to model. In fact if we know $\mathcal{P}(x)$ it is unnecessary to calculate the posterior because $\mathcal{P}(x)$ itself could describe the position of the subject. If we know $\mathcal{P}(x)$, we could estimate the mean from $\mathcal{P}(x)$ directly. So, it seems we need to compute the posterior

$\mathcal{Q}(\mu, \sigma|x)$ from a function containing an unknown element. In probabilistic modeling, if we cannot predict the density function the best we can do is to assume it as a uniform distribution and make $\mathcal{P}(x)$ constant. The uniform $\mathcal{P}(x)$ assumption is acceptable in Bayesian inference [112]. Therefore, the posterior estimation $\mathcal{Q}(\mu, \sigma|x)$ is computed from Equation (2.30).

$$\mathcal{Q}(\mu, \sigma|x) \propto \mathcal{L}(x|\mu, \sigma) \mathcal{P}(\mu, \sigma) \quad (2.30)$$

In order to visualise the problem, let the prior is represented by a table, where μ and σ are varied by row and column. Each element in the prior table comes from the prior density, *e.g.* uniform distribution. In another table, the table of likelihood function, each element is computed by inserting x , μ and σ into the Gaussian function. Once we have two tables, which express prior and likelihood, the posterior probability is element-wise multiplication of two tables. In order to compute the the expectation of μ and σ , the posterior integration is computed as shown in Equation (2.31) and 2.32.

$$\langle \mu \rangle = \iint \mu \cdot \mathcal{Q}(\mu, \sigma|x) \, d\mu d\sigma \quad (2.31)$$

$$\langle \sigma \rangle = \iint \sigma \cdot \mathcal{Q}(\mu, \sigma|x) \, d\mu d\sigma \quad (2.32)$$

We can estimate the distance by searching a peak from the posterior density or computing the expectation $\langle \mu \rangle$ as shown in Equation (2.31). The posterior may have many peaks in a non-Gaussian problem. The expectation value is computed from integration over μ and σ . It seems difficult to implement integration on a computer because even in this simple case we have 2 dimensional integration for an unknown variable. Computing the high-dimensional integration directly with a standard method such as the midpoint method is slow. However, with the right tool, such as Monte Carlo method, the integration can be computed easily.

Before we move forward we need to realise that the state variables is usually estimated from indirect measurement (the state variables cannot be measure directly). The unobservable state is called *the hidden state*, which needs to be measured indirectly by using the conditional probability and transformation between the state space and the measurable space. In computer vision the observation is a 2D image, whereas the state

space can be a position in the 3D world coordinate systems. So, the measurement involves transformation between the 2D and 3D coordinate system. Transformation from world to image coordinates will be described in Section 2.6.

2.5.1 Monte Carlo methods

Monte Carlo integration [113, 114] or the MC method is a simple idea to perform numerical integration by randomly examining the considered function. A set of samples $x_i \in \mathbb{R}^m$, where $i = 1, 2, \dots$, are drawn from a distribution to cover the considered function boundary. The resulting estimation is as Equation (2.33), where V is a volume of the boundary covering $f(x)$, and V has dimension $m + 1$. N is the number of samples.

$$\int_{x=0}^{x=1} f(x)dx \approx \frac{V}{N} \sum_{i=1}^N f(x_i) \quad ; x \sim \mathcal{U}(0,1) \quad (2.33)$$

For example, in Figure 2.4 we would like to calculate an integration of a function $f(x) = \sqrt{1 - x^2}$ from $x = 0$ to $x = 1$ by the MC method. First we have to draw many samples of x_i from a uniform distribution, which generates samples uniformly in a square region from $x = 0$ to $x = 1$ and from $y = 0$ to $y = 1$. We then compute the summation of $f(x)$. The area V in this case is unity. The symbol \sim describes that a set of samples $x = \{x_i; i = 1, 2, \dots, N\}$ are drawn from a uniform distribution in range $x = 0$ to $x = 1$ and the uniform distribution is denoted by $\mathcal{U}(0,1)$.

When m (dimension of x) is high the MC method is relatively faster than the mid-point or the trapezoid integration method [115] (page 443). The standard mid-point integration method has the number of summation operations (summing all elements) equal to the dimension m , and the computational complexity increase exponentially relative to the dimension growth. In contrast, the MC method has a single summation operator. In terms of convergence, the error of the MC estimation is proportional to the inverse of the square root of N . When the size of samples N is large enough, the error converges to zero as in Equation (2.34), where $\text{var} \langle f(x) \rangle$ is a variance of the function $f(x)$.

$$\text{error} = V \sqrt{\frac{\text{var} \langle f(x) \rangle}{N}} \quad (2.34)$$

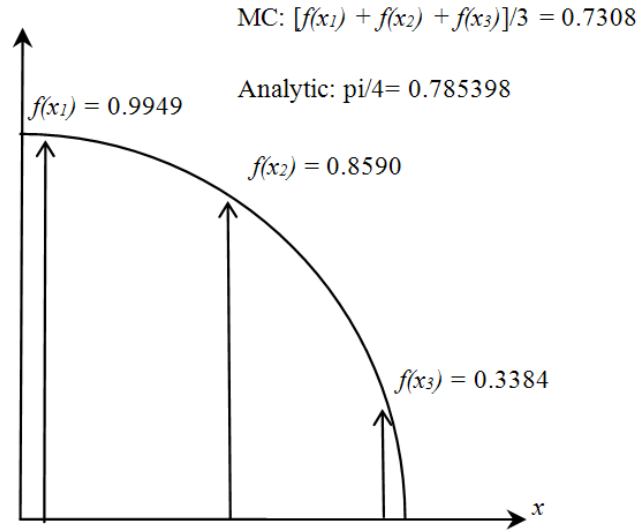


FIGURE 2.4: An approximation of $\int_{x=0}^{x=1} \sqrt{1-x^2} dx$ by MC integration method

The MC method requires a good boundary approximation. The boundary must cover the considered density function. In the case of a very broad distribution, the probability density function usually has large areas of low density. It is worthless to sample in a region which has density near zero, because sampling at a low density location makes a tiny increment to the integration. Therefore, it is inefficient to sample the function around the low density regions.

2.5.1.1 Importance sampling

The *importance sampling* method, *e.g.* [116], has been proposed to solve wasteful uniform sampling. The key idea of importance sampling is to emphasis sampling in high density areas. Even though the subject function is unable to be known exactly, we can manage to get a good guess. This prediction may need more information depending on the problem domain. Once we can estimate a subject function $f(x)$ by $g(x)$, we can estimate the integration by using importance sampling.

$$\int f(x) dx = \int \frac{f(x)}{g(x)} g(x) dx \quad (2.35)$$

$$= \frac{V_g}{N} \sum_{i=1}^N \frac{f(x_i)}{g(x_i)} \quad ; x_i \sim g(x) \quad (2.36)$$

where the V_g is the volume under function $g(x)$. In general V_g is normalised to 1 for every probability density function. Transforming from integration to summation by sampling from density function $g(x)$ allows us to remove $g(x)$ from the integration.

An integration of a multiplication between two functions is called *the inner product*. An inner product between a prior and likelihood is a posterior density function as shown in the beginning of this section. From the importance sampling technique, the inner product of two functions can be formed by summing $f(x)$ and drawing samples from $g(x)$.

$$\int f(x)g(x)dx = \frac{1}{N} \sum_{i=1}^N f(x_i) \quad ; x_i \sim g(x) \quad (2.37)$$

As an example from Equation (2.31), we can use importance sampling to estimate an expectation value by substituting the likelihood for $f(x)$ and the prior for $g(x)$. Since we know the prior density, we can calculate the expectation value from Equation (2.40).

$$\langle \mu \rangle = \iint \mu \mathcal{Q}(\mu, \sigma | x) d\mu d\sigma \quad (2.38)$$

$$\langle \mu \rangle \propto \iint \mu \mathcal{L}(x | \mu, \sigma) \mathcal{P}(\mu, \sigma) d\mu d\sigma dx \quad (2.39)$$

$$\langle \mu \rangle \propto \frac{1}{N} \sum_{i=1}^N \mu_i \mathcal{L}(x_i | \mu_i, \sigma_i) \quad ; (\mu_i, \sigma_i) \sim \mathcal{P}(\mu, \sigma) \quad (2.40)$$

For measurement x_i , the prior parameters (μ_i, σ_i) are sampled proportional to prior density $\mathcal{P}(\mu, \sigma)$ and the expectation is estimated as in Equation (2.40). In order to draw samples according to an arbitrary prior density, we can use rejection sampling or inverse transform sampling.

2.5.1.2 Rejection sampling

In 1951, John Von Neumann used *rejection sampling* [117] to generate samples for Monte Carlo methods. A key idea of rejection sampling is producing a sample x from a standard random generator such as uniform or normal distribution then rejecting some samples with a probability which is proportional to the rejecting-to-accepting ratio, see Figure 2.5. Note that the x is an unknown variable of a function not a measurement from the previous example.

The distribution $g(x)$ is chosen from a standard random generator and multiplied by an arbitrary constant C to make $C.g(X)$ cover $f(x)$. First we sample x_o from density $g(x)$. When we draw a vertical line at x_o we can calculate $f(x_o)$ and the envelope $C.g(x_o)$. Then a random variable u is generated from another uniform random generator $\mathcal{U}(0,1)$. If the height ratio of $f(x_o)/C.g(x_o)$ is larger than u , the sample x_o will be accepted, otherwise x_o will be rejected and regenerated until the condition is met.

The rejection method generates many redundant samples and many samples are rejected. So this method is inefficient.

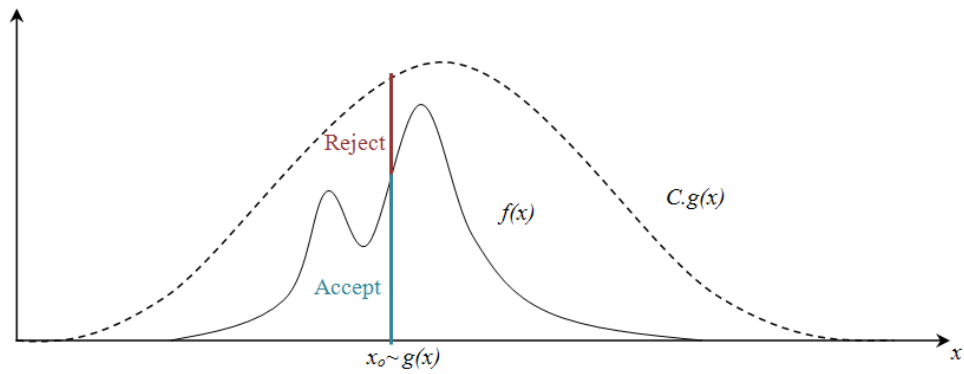


FIGURE 2.5: Generating an sample by Rejection sampling method

2.5.1.3 Inversion sampling

In 1986, Luc Devroye used *inversion sampling* [118] to generate samples from any particular density function. For any particular probability density function $f(x)$ we can calculate the cumulative density function $F(x)$, which is monotonically increasing up to 1. We can transform a uniform distribution to $f(x)$ by using the inverse function of the cumulative function as Equation (2.41) and 2.42. Figure 2.6 shows the inversion sampling process.

$$x \sim f(x) \quad (2.41)$$

$$x = F^{-1}(u) \quad ; u \sim \mathcal{U}(0,1) \quad (2.42)$$

This method has no rejection, so it can generate samples efficiently. The inversion method has been used in the SIR particle filter [119] in the re-sampling step.

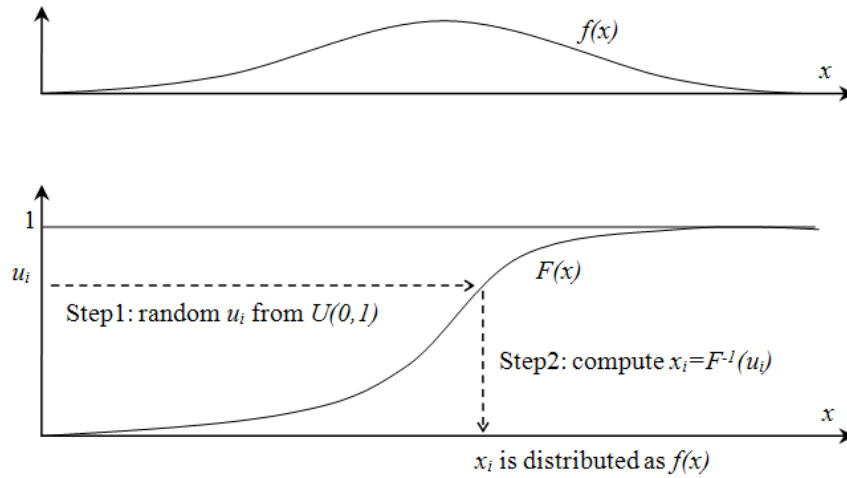


FIGURE 2.6: Inversion sampling transforms a uniform distribution to the objective distribution $f(x)$

2.5.2 Markov Chain Monte Carlo methods

In 1970, Hasting [120] suggested the Markov-chain model could be applied to MC integration. This is a Markov Chain Monte Carlo (MCMC) sampling method. MCMC emphasises how to produce a series of samples, which is called *a chain*. In the chain, the next sample is dependent on the previous sample in the series. MCMC combines MC with a sample chain generator. The chain is automatically produced to explore a high dimensional state space without human intervention.

From the importance sampling method, a large portion of samples should fall in a high density area. To achieve the importance sampling strategy, the MCMC sample generator constructs a proposal density, which is similar to the envelope function in the rejection sampling method, and generates a sample by the rejection sampling method. The rejection sampling method in MCMC is also called *the Metropolis-Hastings algorithm*. The algorithm makes a chain of samples populated around a high density region.

In a high dimensional state space, a landscape of a density function usually consists of several peaks and many low density regions. The chain is likely to be trapped in a peak and unable to jump to other peaks because the Metropolis-Hastings algorithm rejects all low probability sampling. So, it is unlikely to get good sampling in the desired direction. Almost all of the samples will be rejected when the chain attempts to move

across the low density region between peaks. The chain is trapped in the same region and unable to explore new dense areas.

To solve the problem we have to reduce number of choices (dimension) and consider one particular dimension for each sample. Moving in a single dimension is more likely to get the dense region because of a lower number of choices. Searching in one dimension reduces the search space exponentially compared to searching directly through a joint density in higher dimension. The technique that considers one-by-one dimensions for each step is also known as *Gibbs sampling*.

The success of MCMC relies on a sampling method that explores over dense areas and has a small chance to visit low density regions. Furthermore the chain transition has to be unbiased in the time domain, which means that the probabilities of a sample propagating forward in time to next position and back to the original are equal. This balance transition, also called *detailed balance*, is a key for MCMC to reach an equilibrium state [121]. Normally the chain initially needs some period to explore the entire landscape to make the sampling satisfy the detailed balance condition. The initial exploration period is called *the burn-in period*.

To extend MCMC to generate samples with varying dimension, the dimension of the state vector itself may be unknown, for example, estimating the number of clusters in dataset [122] or determining the number of peaks in a LIDAR signal [123]. In these cases, the MCMC sample chain has to leap from \mathbb{R}^n space to \mathbb{R}^{n-1} or \mathbb{R}^{n+1} . In 1995 Peter Green [124] proposed an algorithm designed for jumping between dimensions called Reversible Jump Markov Chain Monte Carlo (RJMCMC). The transition of the chain in this method is in balance, which means the possibility of reducing and increasing the dimension are equal. So the equilibrium of sampling can express a true density function. RJMCMC methods are very useful to examine statistical interference when a number of parameters is varying.

Drawbacks of MCMC Those variation of MCMC methods can investigate density function even in the case of varying the number of dimensions. However, there are at least two significant drawbacks: the chain sampling and sensitivity in initialization.

Parallel MCMC In real-time application such as tracking, the parallel MCMC is desired. The sequential process of the chain sampling makes a parallel implementation difficult. The MCMC chain must be split into many chains and processed in parallel. Splitting the chain may cause dependencies between chains, where samples from different chains are accidentally equal and make two chains have the same trace. The random generator performing on each processor could generate overlapping chain. Parallel chain MCMC has been discussed in [125, 126] and the authors emphasised random generation on each processor. If we could generate random sequences, which make the numbers in all sequences distributed uniformly, we could program MCMC in a parallel fashion. Quasi-random sequences such as [127, 128] can be applied in MC integration effectively because it produces uniformly distributed samples. They are called *sequenced* because the new number is computed from the previous number and we can also perform *the skip-ahead* of the sequence. Imagine we have a random sequence with size N and we have two identical processing cores. Some random seed number is used for generating the first partition of sequence in the first processor. For the second processor, a seed number is set to the half way ($N/2$) in the sequence. With the skip-ahead method it can be guaranteed that numbers in the two sequences are uniform and the chains never overlap. The skip-ahead function in the CURAND library [129] is one example to show that it is possible to implement skipping in the random sequence.

The ideal of multi-chain MCMC is similar to the particle filter which will be discussed in Section 2.5.3. In a particle filter at a particular time, many samples are generated and tested simultaneously.

2.5.3 SIR Particle filter

The particle filter represents the prior (or the posterior) density by a set of many points. The points are distributed in state space according to the density function. The points are similar to particles in space so it is called *the particle filter*. There are many variations in the particle filter family and they have been influenced by the MC technique.

The Sequential Importance Resampling (SIR) particle filter was introduced in [119] and applied to visual tracking in [23]. The SIR particle filter is well known because the sampling strategy is effective. The particle filter is also called a Sequential Monte Carlo method because it processes a series of observations but the samples are generated in

parallel. Instead of using the single chain of samples the SIR particle filter produces a lot of samples at the same time. In every frame an observation is fed to the particle filter and parameters are estimated sequentially in time domain. No record of observations is saved and in each frame only a current observation is considered. The word “sequential” comes from processing a series of observations. However, it is possible to implement the sampling in the particle filter in a parallel version because all particles are generated at the same time.

To describe the SIR PF algorithm let us consider the following example. Assume we are going to measure the state of a moving car by using a Sonar instrument, which can return distance X between the object and the observer every period Δt .

State variables Let X be the measurement distance and we need to recursively estimate the position and velocity. To separate the noisy measurement X from estimation state, we represent the estimation of state by expectations of position and velocity $S = (\langle X \rangle, \langle U \rangle)$.

Transition functions the dynamic of the car is modeled by two simple equations of motion as shown in Equation (2.43) and Equation (2.44). These state transformations in time domain are called *transition functions*. The noise \mathcal{E} is drawn from a Gaussian distribution. The dynamic model assumes that in a small interval of time the deterministic car position depends only on velocity. The noise in the velocity equation represents unknown acceleration, which makes the velocity state dispersed in state space.

$$\langle X \rangle_t = \langle X \rangle_{t-1} + \langle U \rangle_{t-1} \cdot \Delta t + \mathcal{E}_X \quad ; \mathcal{E}_X \sim \sigma_X \cdot \sqrt{\Delta t} \cdot \mathcal{N}(0, 1) \quad (2.43)$$

$$\langle U \rangle_t = \langle U \rangle_{t-1} + \mathcal{E}_U \quad ; \mathcal{E}_U \sim \sigma_U \cdot \sqrt{\Delta t} \cdot \mathcal{N}(0, 1) \quad (2.44)$$

Prior density Once we have the dynamic model we can construct the prior density from the stochastic transition functions. In general, to reduce computational complexity the current state $(\langle X \rangle_t, \langle U \rangle_t)$ is modeled to depend on only the previous state $(\langle X \rangle_{t-1}, \langle U \rangle_{t-1})$ as a first-order Markov chain. When the noise is added to the model, the system has been changed from a linear deterministic system to a stochastic system. The state variables are now represented by a distribution in joint state space. The distribution of the state variables is prediction and is also called *a priori density*. In order

to express the density function, the particle filter uses many of distributed state vectors, $\mathcal{P}(S) = \{S_n; n = 1, 2, \dots\}$. Single state S_n is also called a particle state with a particle index n .

Likelihood function Each particle state is evaluated. If we assume the the measurement distributes as a Gaussian, the likelihood is a probability of the measurement occurring given the distribution. The likelihood is computed from the probability function and the measurement as in Equation (2.45). Note that in order to simplify the problem, the variance is excluded from the state variables (unlike Equation (2.29)).

$$\mathcal{L}(X|\langle X \rangle, \langle U \rangle) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{X-\langle X \rangle}{\sigma})^2} \quad (2.45)$$

Posterior density In order to represent the posterior, each particle in the prior density is multiplied by the likelihood individually. The prior particles have the same weight but the posterior particles have different weights according to the likelihood function. So the posterior density is represented by a set of weighted particles in the state space.

Re-sampling To prevent a particle from spreading to a low probability region, the re-sampling method is applied as suggested in the importance sampling method, Section 2.5.1.3. Then the process is repeated by transforming the particles to the next iteration using the transition function.

Figure 2.7 shows the process of the Particle filter. A prior density $\mathcal{P}(S)$ is represented by a set of particles. Then the likelihood $\mathcal{L}(X|S)$ of each particle is evaluated. The resulting likelihood of a particle S_n is denoted by the weight w_n . A mass distribution of the particles is computed from the position of the particles and their weight. The mass distribution is an approximation of a posterior density. Particles for the next iteration are produced from the posterior by inversion sampling described in Section 2.5.1.3. Then particles are transformed by a dynamic model in Equation (2.43) and 2.44. After applying the dynamic model all weights will be reset and the particles will represent the prior density of the next frame.

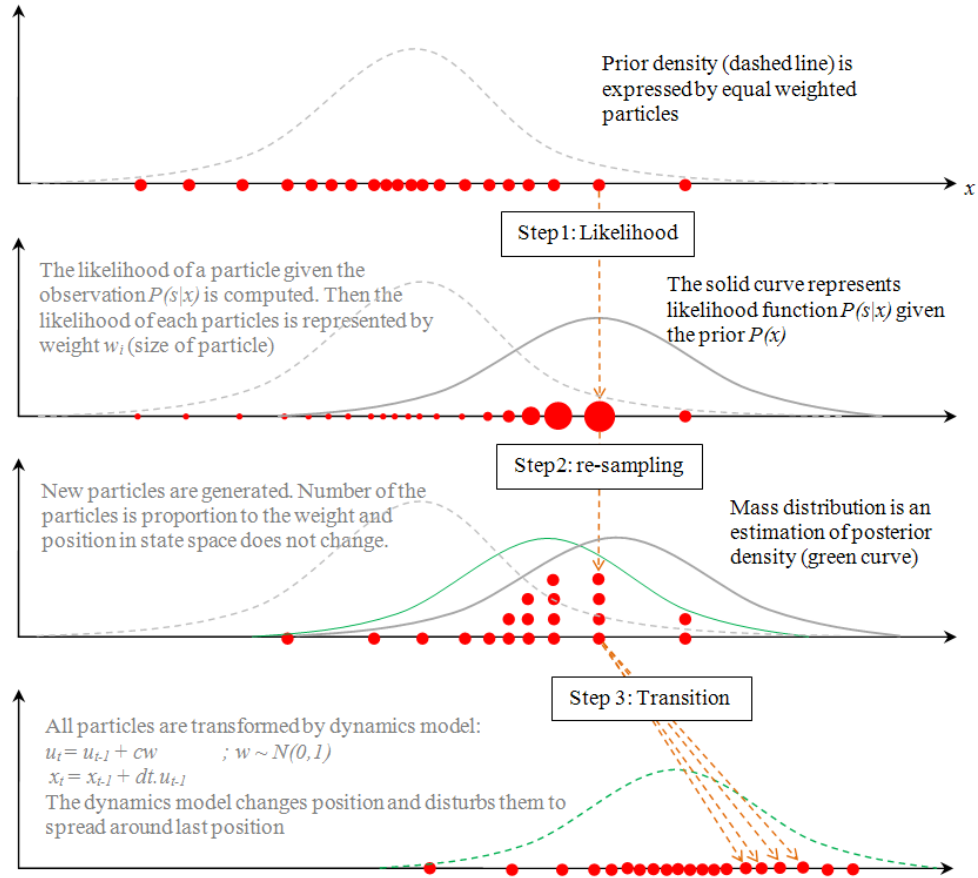


FIGURE 2.7: Three steps of SIR particle filter algorithm from top to bottom: likelihood evaluation, re-sampling and dynamics transition.

We can also use a particle filter to estimate the discrete state. In the case of a discrete state model, we can apply a Markov chain model to regenerate the discrete state instead of the dynamic motion model.

The likelihood computation is an interface between observations and the prior density in a Bayesian framework. We can apply a particle filter to any problem as long as we can model the transition and likelihood functions. The likelihood definition in [111] needs probabilistic modeling. Probabilistic modeling of subject appearance is difficult and can also have a normalising problem. Thus, the likelihood is computed by similarity between an observation image and a synthetic image (image generated from the state and the appearance descriptor).

2.5.4 Likelihood and Similarity measure

Harris's likelihood [111] is the probability of occurrence of an configuration $\mathbf{x} = \{x_1, x_2, \dots, x_N\}$ given prior density $g(x; \theta)$, where θ is a set of parameters to characterise the density shape.

For example, the prior density is a normal density $g(x; \mu, \sigma)$ with mean μ and standard deviation σ . Measurements produce an observation configuration $\{x_1, x_2, \dots, x_N\}$. Then we can determine the likelihood as a series of products of $g(x_i; \mu, \sigma)$ for every x_i .

$$\mathcal{L}(\mathbf{x}|\mu, \sigma) = C \prod_{i=1}^N g(x_i; \mu, \sigma) \quad (2.46)$$

This likelihood definition is directly derived from probabilistic theory. However, we have a problem with the normalising factor C . The likelihood is a density function, so integral over the prior parameter space $\{\mu, \sigma\}$ should be 1. However, the normalising factor C depends on the number of measurements N and it is really important in order to deal with the case of varying N observations. The likelihood function also depends on nuisance parameters, such as the variances of each state. These cause normalizing the likelihood function to be difficult. The likelihood is usually unnormalized so it shouldn't be called as a probability density.

Therefore in many visual tracking implementations, the likelihood is approximated by similarity or dissimilarity measures without concerning the normalising factor [24, 34, 81]. The similarity function is non-negative everywhere. To compute the similarity between an observation image and a synthetic image, we need to define the data structure of an image first.

An image consists of many pixels arranged in horizontal (u) and vertical (v) directions. Each pixel is denoted by $\mathbf{I}(p_u, p_v)$, where $\{p_u, p_v\}$ is a pixel coordinate. A particular pixel $\mathbf{I}(p_u, p_v)$ may have a vector or a scalar value. For example, a pixel constructed from red green and blue channels, $\mathbf{I}(p_u, p_v) = \{r, g, b\} \in \mathbb{R}^3$. The colour vectors $\{r, g, b\}$ is in a space field ($\mathbf{p} = \{p_u, p_v\}$). The vector could be any other form such as a gradient value, $\mathbf{I}(p_u, p_v) = \{\frac{dI^O}{du}, \frac{dI^O}{dv}\} \in \mathbb{R}^2$, computed from a gray scale image I^O .

The vector array of image $\mathbf{I}(p_u, p_v)$ can be such histogram, a Haar feature, optical flow *etc.* An image is constructed from a vector array $\mathbf{I}(p_u, p_v)$ in 2D space $\mathbf{p} = (p_u, p_v)$. The dimension of a pixel vector can be large as $\mathbf{I}(\mathbf{p}) = \{I_i; i = 1, 2, \dots, i_{max}\}$.

So, one possible method to compute similarity is the product of the exponential of dissimilarity [81, 130]. For example, the likelihood between an observation image $\mathbf{I}^A(\mathbf{p}^A)$ and a synthetic image $\mathbf{I}^B(\mathbf{p}^B; \theta)$ generated by prior parameter θ could be computed from the product sequence as Equation (2.47).

$$\text{similarity} = \prod_{\mathbf{p}=\mathbf{p}^A \cap \mathbf{p}^B} \exp \{ -\text{dis} [\mathbf{I}^A(\mathbf{p}), \mathbf{I}^B(\mathbf{p}; \theta)] \} \quad (2.47)$$

2.5.5 Summary

The Bayesian estimator calculates the posterior density of the state from the prior density and likelihood function. The numerical estimation involves generating random samples according to the prior density. In the particle filter, the prior density is represented by a set of particles, which allow the process to be made parallel. One fact we need to keep in mind is that the numerical method of the estimator is highly complicated due to the huge number of samples required. It is a great challenge to implement the method in real-time. In visual tracking, computing the likelihood function is a relatively slow process due to large input data.

2.6 Camera Calibration

In order to perform tracking in a multi-camera network, all subject positions must be described in the same coordinate system. If we cannot unify the representation of the subjects in state space, integrating observations from many cameras is impossible. In this work, tracking by multiple cameras always describes the subject state in the exact location of a real world coordinate system rather than in an image coordinate system. The camera calibration unifies all camera coordinate systems into a single system. The calibration also allows us to use the ray-tracing method to generate a synthetic image in order to compute the likelihood function.

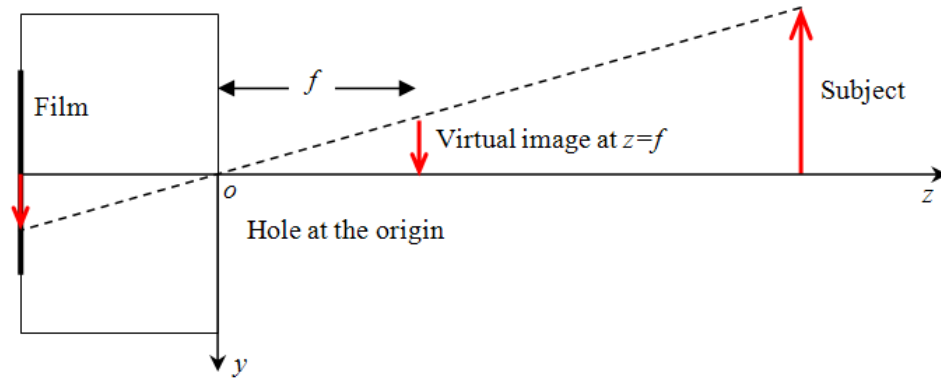


FIGURE 2.8: A pinhole camera model

Consider Figure 2.8 which is a basic pinhole camera model of a light ray from a surface of an object projected on to a film or a sensor. The sensor surface is also called *the image plane*. The pinhole camera consists of a small hole at the origin o that allows light rays pass to the image plane. The distance between the hole and image plane is denoted by f . Modern cameras have optical lenses in order to condense more light to the low sensitive sensor. The large optical lenses make modern cameras able to work in a dark environment. In modern cameras the distance f is approximately equivalent to the distance between the image plane and the centre of the lens. The lenses can bring about barrel and pincushion distortion to a projected image. Barrel distortion is normally found in commercial lenses. They project a perfect square to be bent out on the edges similar to a barrel shape. For narrow lenses, such as a lens with less than 30 degree of view on the diagonal, the distortion is negligible. The classic pinhole camera model has no distortion as the light goes in a straight line.

Figure 2.9 shows a transformation between point \mathbf{q} and \mathbf{p} , where the distance between the red plane and origin is unity because the focal length is normalised. The fixed plane $z = 1$ represents the image plane and the coordinate system (u, v) is a pixel coordinate on the image plane. All light rays must pass through the origin o so it is equivalent to an aperture of the pinhole camera. This model can also describe a narrow view modern camera, where the distortion is very small.

In [131], Hartley suggested that Tsai's camera model [132] can be described in matrix form. Here the 3-by-4 matrix is known as a projection matrix P , which transforms a point \mathbf{q} in a 3D world coordinate system to a point \mathbf{p} in a 2D coordinate system on an image plane. The projected coordinate on the image plane $\mathbf{p} = [p_u \ p_v]^T$ is expressed

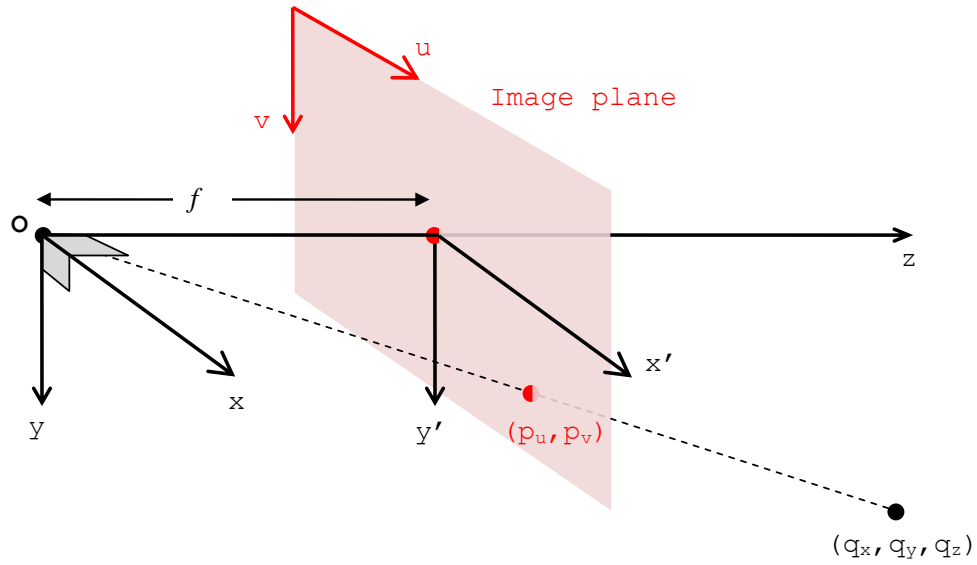


FIGURE 2.9: Projection from a 3D point (q_x, q_y, q_z) to a 2D point (p_u, p_v) on an image plane at $z = 1$. Note that translation and rotation are removed to simplify the model.

by homogenous coordinates. The homogenous coordinate system reduces the vector dimension by normalising the third element to a unit. p_w is pointing to the same direction as z and always normalised to 1 to represent the fixed location of image plane in z direction. Therefore, the p_w cannot be represented on the image plane. Note that the vector \mathbf{p} has 3 elements to represent a line of a light ray and is projected to the image plane by normalising $\mathbf{p} = 1$, which is an intersection between the ray and the image plane.

$$\begin{bmatrix} p_u \\ p_v \\ p_w \end{bmatrix} = \begin{bmatrix} \alpha & \gamma & \epsilon_u \\ 0 & \beta & \epsilon_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_1 \\ R_{21} & R_{22} & R_{23} & t_2 \\ R_{31} & R_{32} & R_{33} & t_3 \end{bmatrix} \begin{bmatrix} q_x \\ q_y \\ q_z \\ 1 \end{bmatrix} \quad (2.48)$$

The relation of \mathbf{p} and \mathbf{q} is described by a linear system in Equation (2.48). The projection matrix \mathbf{P} is a product between an intrinsic matrix \mathbf{K} and extrinsic matrix $[\mathbf{R}, \mathbf{t}]$. The intrinsic matrix \mathbf{K} describes an internal property of the camera and an extrinsic matrix $[\mathbf{R}, \mathbf{t}]$ explains the translation and orientation of the camera relative to the reference origin point. It means moving or rotating the camera change $[\mathbf{R}, \mathbf{t}]$ but does not affect \mathbf{K} . The matrix \mathbf{P} has 11 unknown parameters inherited from \mathbf{K} and $[\mathbf{R}, \mathbf{t}]$ matrices. In order to calibrate \mathbf{P} , we need to know $\alpha, \beta, \gamma, \epsilon_u, \epsilon_v, t_1, t_2, t_3$ and

three angles of rotation matrix.

$$\mathbf{p} = \mathbf{K}[\mathbf{R}, \mathbf{t}]\mathbf{q} \quad (2.49)$$

$$\mathbf{p} = \mathbf{P}\mathbf{q} \quad (2.50)$$

It is easy to work with in pixel units so the intrinsic parameters $(\alpha, \beta, \gamma, \epsilon_u, \epsilon_v)$ are converted from standard length to pixel units. Therefore, the focal lengths α and β are in pixel units. Spacing between two neighbour photo-receptors in u and v directions are denoted by d_u and d_v . In some cameras d_u and d_v are unequal. The focal lengths in pixel units are shown in Equation (2.51) and eq focal beta.

$$\alpha = \frac{f}{d_u} \quad (2.51)$$

$$\beta = \frac{f}{d_v} \quad (2.52)$$

A camera direction z passing from origin o intersects the image plane at the image centre (ϵ_u, ϵ_v) , as shown in Figure 2.10. The parameters ϵ_u and ϵ_v are the centre point of the camera in pixel units (the normal vector pointing out from image plane at pixel (ϵ_u, ϵ_v) is exactly directed to the camera centre o). The parameter γ describes skewness between pixel axes (u, v) on an image plane. Typically, $\gamma = 0$, which means u and v are exactly perpendicular.

Methods to estimate intrinsic and extrinsic parameters are known as camera calibration. In 1987, Tsai [132] developed coplanar calibration method, which is very flexible and easy to replicate, and later Zhang [133] simplified and popularised Tsai's method. A flat chessboard, which can easily be produced by a normal printer, was used in the method. Compared to non-coplanar calibration it uses a 3D object such as a cube and the method faces a difficulty to produce a 3D calibration object which can easily be deformed. In the case of using a chessboard, we can draw many squares on it and place it on a flat floor, which is quite simple. The coplanar method has become popular in recent research [134–137].

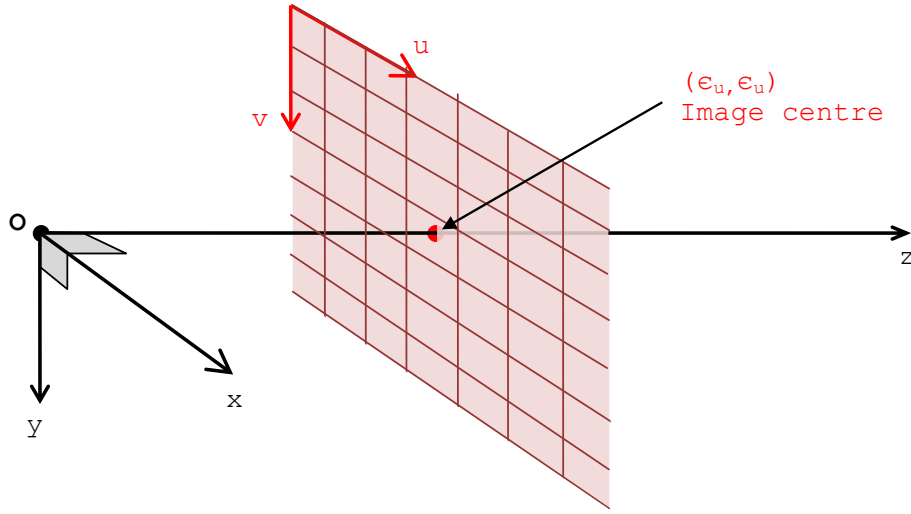


FIGURE 2.10: Relation between 3D coordinate system (x, y, z) and pixel coordinate system (u, v) .

2.6.1 Homography

To make this thesis self-contained, I will briefly describe Zhang's method [133]. In the coplanar calibration method the feature points in 3D are on a flat plane. To simplify the problem all corner points are set at $z = 0$. A point \mathbf{q} is a corner on a chessboard and point \mathbf{p} is a projected corner in a pixel coordinate system. The setting makes $q_z = 0$. So, we can remove the third column of the rotation matrix from the Equation (2.48).

$$\begin{bmatrix} p_u \\ p_v \\ p_w \end{bmatrix} = \begin{bmatrix} \alpha & s & \epsilon_u \\ 0 & \beta & \epsilon_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & t_1 \\ R_{21} & R_{22} & t_2 \\ R_{31} & R_{32} & t_3 \end{bmatrix} \begin{bmatrix} q_x \\ q_y \\ 1 \end{bmatrix} \quad (2.53)$$

A 3-by-3 matrix that transforms a point on a flat plane to another flat plane is called a *homography matrix* \mathbf{H} [131]. The product $\mathbf{K}[\mathbf{R}, \mathbf{t}]$ is a homography as all points on a chessboard are in the same flat plane. In order to solve all 11 camera parameters, we need at least 11 independent equations. A pair of projecting points from the chessboard to the image plane gives 2 equations in the u and v components. In order to compute a homography matrix, which contains 8 unknowns, we need at least 4 pairs of points. At least 3 homography matrices must be constructed before determining the 11 unknown camera parameters.

$$\mathbf{p} = \lambda \mathbf{H} \mathbf{q} \quad (2.54)$$

$$\mathbf{H} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} = \begin{bmatrix} \mathbf{h}_1^T \\ \mathbf{h}_2^T \\ \mathbf{h}_3^T \end{bmatrix} \quad (2.55)$$

A homography matrix \mathbf{H} transforms \mathbf{q} to \mathbf{p} , where \mathbf{h}_i^T is a row vector in \mathbf{H} and λ is a homogeneous normalising factor. \mathbf{H} has 8 unknowns and 1 normalising factor. For each pair of points (\mathbf{p} and \mathbf{q}) we can produce 2 equations. Therefore we need at least 4 pairs of \mathbf{p} and \mathbf{q} in order to get an exact solution of matrix H .

The method to solve \mathbf{H} is known as direct linear transformation (DLT) [131, 132]. Because \mathbf{p} is a homogeneous coordinate, therefore, $\mathbf{p} - \mathbf{H}\mathbf{q} \neq \mathbf{0}$. However, \mathbf{p} and $\mathbf{H}\mathbf{q}$ are parallel, so their cross product is zero.

$$\mathbf{0} = \mathbf{p} \times \mathbf{H}\mathbf{q} \quad (2.56)$$

$$\mathbf{0} = \begin{bmatrix} p_v \mathbf{h}_3^T \mathbf{q} - p_w \mathbf{h}_2^T \mathbf{q} \\ p_w \mathbf{h}_1^T \mathbf{q} - p_u \mathbf{h}_3^T \mathbf{q} \\ p_u \mathbf{h}_2^T \mathbf{q} - p_v \mathbf{h}_1^T \mathbf{q} \end{bmatrix} \quad (2.57)$$

Then we can rearrange the equation to a linear equation system $\mathbf{M}\mathbf{h} = \mathbf{0}$, where

$$\mathbf{M} = \begin{bmatrix} 0 & 0 & 0 & -p_w q_x & -p_w q_y & -p_w & p_v q_x & p_v q_y & p_v \\ p_w q_x & p_w q_y & p_w & 0 & 0 & 0 & -p_u q_x & -p_u q_y & -p_u \end{bmatrix} \quad (2.58)$$

and

$$\mathbf{h} = \begin{bmatrix} H_{11} & H_{12} & H_{13} & H_{21} & H_{22} & H_{23} & H_{31} & H_{32} & H_{33} \end{bmatrix}^T. \quad (2.59)$$

The 2-by-9 matrix \mathbf{M} is constructed from a pair of \mathbf{p} and \mathbf{q} . Note that p_w and H_{33} are normalised to 1 by λ . If we capture an image, which has 4 different pairs of \mathbf{p} and \mathbf{q} , we can construct 8 rows of matrix \mathbf{M} and able to solve \mathbf{H} . It means the chessboard must have at least 4 landmark points. However, in practice many landmark points are used to reduce error. In the case the the number of equations is larger than the number of unknowns it is known as *an over-determined system*. Singular value decomposition

(SVD) is applied to solve over-determined system equation $\mathbf{M}\mathbf{h}=\mathbf{0}$. Therefore, we can calculate a Homography \mathbf{H} from a chessboard image.

In order to solve 11 camera parameters many Homography matrices are used.

2.6.2 Camera calibration method

In Zhang's camera calibration method [133], the procedure starts by expressing a coplanar projection (Equation (2.53)) by the homography matrix.

$$\begin{bmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3 \end{bmatrix} = \lambda \mathbf{K} \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix} \quad (2.60)$$

From Equation (2.60), the column vectors of the rotation matrix are

$$\mathbf{r}_1 = \frac{1}{\lambda} \mathbf{K}^{-1} \mathbf{h}_1 \quad (2.61)$$

$$\mathbf{r}_2 = \frac{1}{\lambda} \mathbf{K}^{-1} \mathbf{h}_2. \quad (2.62)$$

Using the fact that columns of the rotation matrix are orthogonal to each other.

$$\mathbf{r}_1^T \mathbf{r}_2 = 0 \quad (2.63)$$

$$\mathbf{r}_1^T \mathbf{r}_1 - \mathbf{r}_2^T \mathbf{r}_2 = 0 \quad (2.64)$$

Then, substituting the rotation vectors.

$$\mathbf{h}_1 \mathbf{K}^{-T} \mathbf{K}^{-1} \mathbf{h}_2 = 0 \quad (2.65)$$

$$\mathbf{h}_1 \mathbf{K}^{-T} \mathbf{K}^{-1} \mathbf{h}_1 - \mathbf{h}_2 \mathbf{K}^{-T} \mathbf{K}^{-1} \mathbf{h}_2 = 0 \quad (2.66)$$

Let $\mathbf{B} = \mathbf{K}^{-T} \mathbf{K}^{-1}$. Note that \mathbf{B} is a symmetric matrix which has 6 unknown parameters. From the equations Equation (2.65) and 2.66, Zhang constructed a linear equation system $\mathbf{A}\mathbf{b} = \mathbf{0}$ (\mathbf{b} is unknown).

$$\begin{bmatrix} \mathbf{A}_{12} \\ \mathbf{A}_{11} - \mathbf{A}_{22} \end{bmatrix} \mathbf{b} = \mathbf{0} \quad (2.67)$$

where

$$\begin{aligned} \mathbf{b} &= \begin{bmatrix} B_{11} & B_{12} & B_{22} & B_{13} & B_{23} & B_{33} \end{bmatrix} \\ \mathbf{A}_{ij} &= \begin{bmatrix} h_{i1}h_{j1}, & h_{i1}h_{j2} + h_{i2}h_{j1}, & h_{i2}h_{j2}, & h_{i3}h_{j1} + h_{i1}h_{j3}, & h_{i3}h_{j2} + h_{i2}h_{j3}, & h_{i3}h_{j3} \end{bmatrix} \end{aligned} \quad (2.68)$$

So far, from a homography matrix we can generate two rows of matrix \mathbf{A} . In order to solve 6 unknowns in \mathbf{B} , we need 3 independent homography matrices equivalent to 3 images of the chessboard captured by the same camera. Using more images gives a smaller error and again in order to solve an over-determined problem, the SVD method is exploited. As described in [133], once vector \mathbf{b} is computed, all intrinsic parameters can be calculated by using Equations from 2.69 to 2.74.

$$\epsilon_v = \frac{B_{12}B_{13} - B_{11}B_{23}}{B_{11}B_{22} - B_{12}^2} \quad (2.69)$$

$$\lambda = B_{33} - \frac{B_{13}^2 + y_o(B_{12}B_{13} - B_{11}B_{23})}{B_{11}} \quad (2.70)$$

$$\alpha = \sqrt{\frac{\lambda}{B_{11}}} \quad (2.71)$$

$$\beta = \sqrt{\frac{\lambda B_{11}}{B_{11}B_{22} - B_{12}^2}} \quad (2.72)$$

$$s = -\frac{\alpha^2 \beta B_{12}}{\lambda} \quad (2.73)$$

$$\epsilon_u = \frac{s\epsilon_v}{\beta} - \frac{\alpha^2 B_{13}}{\lambda} \quad (2.74)$$

The extrinsic parameters can be computed by Equations 2.75 to 2.78.

$$\mathbf{r}_1 = \lambda \mathbf{K}^{-1} \mathbf{h}_1 \quad (2.75)$$

$$\mathbf{r}_2 = \lambda \mathbf{K}^{-1} \mathbf{h}_2 \quad (2.76)$$

$$\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2 \quad (2.77)$$

$$\mathbf{t} = \lambda \mathbf{K}^{-1} \mathbf{h}_3 \quad (2.78)$$

We can also refine the final solution by any least square error optimisation such as Levenberg-Marquardt optimiser to get a smaller error.

2.7 Hardware comparison

The computational power from the specification of a standard CPU, for example the Intel Core i7 965 (4cores) can provide up to 69GFLOPS [10]. The computational power is limited by the small number of cores, which is insufficient for 3D tracking.

An alternative platform is a Field Programmable Gate Array (FPGA), which provides flexibility and more computation power. For example, the computational power of the Virtex-6 SX475T, a high-end FPGA, was estimated at 450GFLOPS [138]. Another model the Virtex-7 from Xilinx [139] can process up to 5,335 GMAC/s (Giga Multiply-Accumulate Operations per Second). A FPGA is a collection of logic gates and registers, which can be programmed as any kind of digital circuit. A FPGA is very useful in Digital Signal Processing (DSP), which involves basic arithmetic operations such as multiplication and addition. In the 3D tracking problem special functions such as inverse, square root and logarithm are very common. Some projects have used FPGA in visual tracking for example [140, 141] but they were restricted to a 2D feature basis because of difficulty of implementing the special functions. The special functions such division [142, 143], square root [144] and trigonometric functions [145] are difficult to optimise and require a significant number of slices (physical resource on a FPGA). In a FPGA, a complex algorithm such as 3D tracking, which involves the inverse of matrix and trigonometric functions, it is extremely difficult to make the framework from scratch. These functions can be found in a commercial library but implementing and testing them still requires many tools. Designing and implementing a FPGA application requires sophisticated tools and digital circuit design experience. These make implementing 3D tracking on a FPGA difficult. For example, in order to perform 3D tracking, the ray tracing method requires special arithmetic circuits. We can construct the operators in a FPGA but it requires huge number of logic gates to make a FPU as showed in Table 2.1. The Virtex-6 SX475T has 74,400 slices, from Table 1 in [146], (elementary blocks to construct a digital circuit). Table 2.1 shows the resource requirement for each operation, from Table 29 in [147]. This gives an idea of how much we can do with the Virtex-6 SX475T. The ray tracing of a single subject needs up to a million floating point operations (combination of addition, multiplication and dividing). If we do not reuse the hardware resources (slices), we may not have sufficient resources. In short,

implementing 3D tracking in a FPGA may face two major problems; complexity of low level design leading to delay in development and also the physical resource limitation.

Even though the FPGAs showed the best speed performance and highest power efficiency in many evaluations [148, 149], the evaluations were conducted on specific applications, which is unfair for CPU and GPU. In [148], Kestur studied Basic Linear Algebra Subprograms (BLAS), which involved multiplications and additions. In [149], they studied random number generator, where the FPGA used binary operations; in contrast CPU and GPU used special functions for mapping the distribution. These evaluations did not consider the special functions. Implementing the special functions requires large amounts of hardware resource (slices). CPU and GPU architectures dedicate a portion of the area on a silicon chip for implementing these special functions. The unit is called the Special Function Unit (SFU).

TABLE 2.1: The resource requirement to implement floating point operations

Operation	Resource ¹ (slices)
Multiply	692
Add/Subtract	626
Int32-to-Single	224
Single-to-Int32	233
Single-to-Double	68
Compare	19
Divide	1,366
Sqrt	809

For decades ray tracing has been a fundamental technique in Computer Graphics and has been applied in many applications, such as 3D gaming, engineering computer aided design or simple visualization. A special device attached to a standard computer has been designed for rendering millions of pixels in a second and is called a Graphics Processing Unit (GPU). A high-end GPU such as the Nvidia Tesla C2050 [150] or AMD FireStream 9270 [151] can provide computational power beyond a TeraFLOPS, which is much more than our requirement.

Another way for reusing the physical resources is to construct processing units, such as a Von Neumann architecture or a stream processing unit, from those available slices. The MicroBlaze [152] under the trade mark of Xilinx is a processor constructed from logic gates in a FPGA and requires software to control it. In short, processor units are created

¹The Single-Precision (32bit) operations are speed optimised and no DSP slice usage

inside the FPGA and the FPGA acts as a multi-core processor. In order to construct one MicroBlaze core in the Virtex-6 family we need about 300 to 1600 slices[152], excluding other peripherals. Roughly we can build 6 MicroBlaze cores in the Virtex-6 and each core has FPU IEEE 754 compatibility (included addition, multiplication and dividing). The normal CPU and MicroBlaze are slightly different but they are logically identical. Those cores are extensions of Von Neumann processors. So applying MicroBlaze is similar to using a CPU. The difference is just the type of processor. In fact the MicroBlaze operates at around 250MHz, whereas the Intel CPU performs at over 3GHz. It seems we are getting back to multi-core CPU option but this time we have to spend more money on programable silicon hardware and more effort to make our own multi-core architecture, which is already there in the market.

TABLE 2.2: CPU GPU and FPGA comparisons

Hardware	Implementation	Speed Performance	Power efficiency
CPU	Easy	Low	Low
GPU	Medium	Medium to high	Medium
FPGA	Hard	High	High

2.8 Summary

Figure 2.2 shows the components of tracking systems. A tracking system needs an appearance descriptor and a state estimator. A tracking system could be as simple as the mean-shift [75], which consists of a state estimator (maximum-likelihood) and an appearance descriptor (colour histogram). A tracking system may also be made from sophisticated components such as Tracking-Learning-Detection in [26], which is constructed from an appearance descriptor (wavelet templates) and a complex state estimator (a combination of detection, motion estimation and data association). From this study we believe these two components (an appearance descriptor and a state estimator) are necessary in every tracking system.

In single camera (monocular) tracking, the fundamental problems of scale and orientation can be solved by invariant features, such as SIFT [54]. However, SIFT does not work well with a deformable object such a human body. Feature points detection is not suitable for people tracking. People detection at the current state-of-the-art has a detection rate of 97.9% at false alarm 0.01%, which is very promising. However, the

people detection needs data association to make a tracking system. Detection cannot create the trajectory of a person; it needs data association to link several detections into paths. The method is very useful in non-calibrated camera systems. In such systems integrating data between many camera seems impossible. Moreover the people detection usually creates detection parameters from training method. Varying the point of view from side-view to top-view affects the detection performance. Whilst solutions for the challenge of reliable detection are still being developed, we use a Bayesian estimator in this work.

In order to integrate data from different cameras into a single system, the cameras have to be calibrated. A grid base visual hull likelihood such as a probability occupancy map (POM) in [101] is a fast method. POM generates the probability density of people occupying a grid-space on a ground floor. However, its accuracy of position estimation relies on the resolution of grids. Estimation by the grid base is a fast computation including converting a visual hull to an occupancy map on a ground floor, however, the method cannot perceive some useful information such as spatial-pattern and colour.

TABLE 2.3: appearance descriptor comparison

Descriptor	Advantage	Disadvantages
Boundary	-low complexity -fast computation	-unable to keep useful features -the boundary is not always obvious, requires segmentation method
Spatial pattern	-keep only rich detailed features by removing low detailed data	-inconsistent with different camera orientations
Adaptive image	-high detail description	-highly complex -uses large memory to describe a subject
Colour	-captures true physical property of the object surface	-illumination sensitive -requires white balance correction
3D model	-model consistent with multiple cameras	-complex likelihood function

Table 2.3 shows a comparison between different appearance descriptors. The non-3D descriptors (boundary, spatial pattern, adaptive image, colour) are defined without considering the camera model. They typically do not require camera calibration and are very useful in a mobile platform, where the camera model changes rapidly. The boundary is the simplest form of an appearance descriptor, which has low complexity and is a fast

descriptor. Extracting the boundary of the objects typically is not straightforward and requires a sophisticated segmentation method in order to get an accurate boundary. The boundary extraction requires more information about the environment to judge whether a pixel is on the subject or is a static obstruction. Obtaining an accurate boundary is difficult due to the surrounding object and constantly changing shape of peoples bodies. However, the approximation of shape, such as parametric shapes, region and silhouette, can be very useful as described in Section 2.2.1.1.

An adaptive image descriptor keeps many details from observation images which causes the computational complexity and memory utilization to be high. Applying this descriptor in multiple target tacking in real-time is limited by the complexity. Furthermore, the adaptive image is always obtained from a camera, but is unable to explain any variation of camera orientation. The image template pool is built to cover the variation of appearances. The implementation of searching a large template pool is complex.

Colour description of subject appearance can capture the actual surface property of the subject. However, estimating the real colour is challenging because the true colour can be estimated after correcting the illumination conditions, such as white-balance and sensitivity in each colour channel.

An image normally consists of large portions of plain and flat areas. These plain areas contribute nothing to the tracking system. In order to remove the plain areas, the spatial pattern uses feature filtering to remove the plain areas and keep only a few feature points. The spatial pattern keeps rich detail with moderate complexity. The drawback of this method is that the spatial feature is defined in the image pixel coordinate system. So, the feature or signature of a subject is expressed in the pixel coordinate system and this make integrating data from different camera orientations significantly difficult.

The non-3D descriptors cannot express the variation of target appearances under different camera orientations. Including the 3D model into the descriptor allows a multiple camera tracking system consistently perceive all observable images. Therefore, the camera calibration and 3D model is at the centre of this project.

The objective of this thesis is to track people in a 3D environment. The adaptive image and spatial pattern are inconsistent when many cameras are used. We will not

use adaptive image and spatial pattern because of the data integration constraint. We are going to use the boundary, colour and the 3D model as a primary observation.

For the state estimator, the maximum likelihood (ML), the maximum a posteriori (MAP), detection and data association are approximations of the Bayesian estimation. In short, the Bayesian estimator is a complete description of these methods and gives better estimation. The reason for using ML and MAP approximations is that Bayesian estimation requires higher computational power. This problem can be solved by choosing a hardware platform that can provide higher computational power. A Bayesian estimator offers two advantages; complete description for probability density and the ability to integrate various form of probabilistic models. In Section 2.5, the numerical Bayesian estimator is discussed in more detail.

2.8.1 Research direction

In this thesis we study the Bayesian tracking method in a calibrated camera network. The Bayesian estimator with a vertex-base 3D model is widely used for tracking but the main drawback is the complexity. The vertex-base body template is a common appearance descriptor in 3D people tracking. The template can be any shape because the vertex template is very flexible. The flexible always comes with the cost that the vertexes projection is a time-consuming task compare to a parametric shape.

Unlike previous methods, we use a parametric ellipsoid model, which makes estimation faster. Comparison between a vertex-base and a parametric-base are discussed in Section 3.3 We apply Bayesian estimation to obtain accurate results compared to the grid base detection method. We exploit the benefit of Bayesian estimation that it can be modeled by various probabilistic models. Thus, we are able to include useful probabilistic models such as membership probability and colour-texture signature into our likelihood model. The parametric ellipsoid model and the texture signature make our tracking novel and reliable.

In order to compare our approach to others, we start with the PETS evaluation [153], which provides dataset and evaluation scores as summarized in Table 2.4 (MOTA= multiple object tracking accuracy). From Table 2.4 a tracking algorithm is constructed from various appearance descriptors and state estimators.

TABLE 2.4: MOTA comparison with results in [153]

Author	MOTA	Appearance description	Estimator
Arsic09 [154]	22%	3D vertex model	Detection[154]
		Detected volume in x,y and time	Data association (Normalized cuts [155])
Berclaz09 [156]	79%	3D visual hull	Detection (POM [101])
		2D positions in ground plane	Data association (LP [156])
Breitenstein09 [157]	75%	Spatial pattern (HOG+ISM)	Detection
		2D positions in image plane	Data association
		2D positions in image plane	Bayesian (Particle filter)

We also implemented our tracking in a GPU to accelerate the computation. From the review above a GPU can improve the speed significantly compared to a CPU. A GPU delivers computational power of around 10 times more than a CPU. GPU implementation is a medium difficulty compared to the low level design of a FPGA implementation, which is very difficult and may cause delay to the research. We decided to implement the tracking framework on the GPU.

The acquired texture is also applied to recognition as described in Chapter 6. The recognition remembers subjects after leaving and re-entering the scene. This shows that cooperation between a tracking and a recognition system can lead to a smarter recognition system.

Chapter 3

Development of Multi-Target Multi-Camera tracking

The chapter starts with a study of single target tracking system in a 3D environment using background segmentation. The primary objective was to perform single target tracking in the 3D environment.

From related work such as [23, 75] performed tracking in 2D, which estimate position of an object in pixel coordinate system. The 2D tracking overlook the camera model and perform tracking in the pixel coordinate system. The 2D tracking use single camera and unable to extent to multi-camera tracking because the 2D tracking done not have geographic information to indicate relationship between cameras.

We included 3D model camera calibration into our single target tracking for two reasons; accuracy and scalability. In real world people motion is limited on the floor and usually in urban environment the floor is flat. We can make a sensible motion of people on a flat floor. The motion model of people on a flat floor is more sensible than motion model on people in pixel coordinate. The 3D tracking include perspective effect as a camera model. We know that two people moving at same velocity can be projected into different velocity in the pixel coordinate. The further subject from camera appears to be slow in image plan. If we do not take the camera model into account, the motion model and other appearances, *e.g.* perspective size and orientation, will be predicted unrealistically. Furthermore, the 3D tracking is able to extend to multi-camera tracking

because the camera model give the geographic information of the cameras. Integrating more cameras into the tracking system can be done systematically.

Tracking in a 3D environment usually uses points-like human template (vertexes) such as [81, 158], which requires hundreds of vertexes to represent an entire human body. In Section 3.2, we are going to present a fast method to generate a human template by using a parametric ellipsoid model. The human body is represented by an ellipsoid surface in 3D, which needs few parameters to process and makes computation light. The ellipsoid also allows us to learn the texture of a human body as it became as a good identity for recognition.

After the first experiment in single target tracking, our objective was extended to multi-target and multi-camera tracking. It was a significant change and much more work was required. The simple single target tracking framework was redesigned for the multiple target tracking problem. New problems of distraction and short-time disappearance will be considered in Section 3.4. To deal with these problems, we design the probabilistic model and distraction suppression mechanism will also be described in Section 3.4. This chapter will describe the algorithm design and our contribution in using and ellipsoid model in tracking method. Implementation and evaluation of the multiple target tracking will be discussed later in Section 4.

3.1 Single person tracking

In this section we consider tracking a single person . In order to provide a gentle introduction to 3D tracking, some practical problems such as occlusion, distraction and detection will be disregarded. The practical obstacles will be considered later in Section 3.4.

This tracking system is based on Bayesian statistics (a state estimator) and 3D ellipsoid (an appearance descriptor). To estimate the state of a subject by posterior estimation, the SIR particle filter[23] is a good candidate because it allows us to model dynamics and appearance of a target. The most interesting property is that the algorithm can be parallelised almost directly.

3.1.1 SIR Particle filter

The Sequential Importance Resampling (SIR) particle filter[23] is in a family of Monte Carlo estimation as described in Section 2.5.3. An objective is to approximate a posterior density function when a prior and a likelihood function are available. To achieve an effective sampling method, the particle filter exploits prediction or a prior density. The prior density is expressed by a set of points in state space, a point is called *a particle*. Then a weight of each particular particle is computed from the likelihood function (or similarity between observation and a synthetic image generated from the particle). The resulting weight and the particle distribution represent a mass distribution of the particle cloud and the mass distribution represents posterior density. A particle in a high density region of the posterior represents high confidence in the state. Once the posterior is computed, the system can determine the output state from the expectation value or the maximum posterior method. In order to prepare the prediction for the next generation, the set of particles representing the posterior density is re-sampled and then transformed by a transition function. An objective of re-sampling is to generate samples proportional to the density because a particle in the dense area will contribute more accuracy as described in Section 2.5.1.1. The transition function can be modeled by the physical and statistical dynamics of the subject. Finally, all weights of particles are equalised so the particle cloud represents a prior density of next generation and the process repeats again.

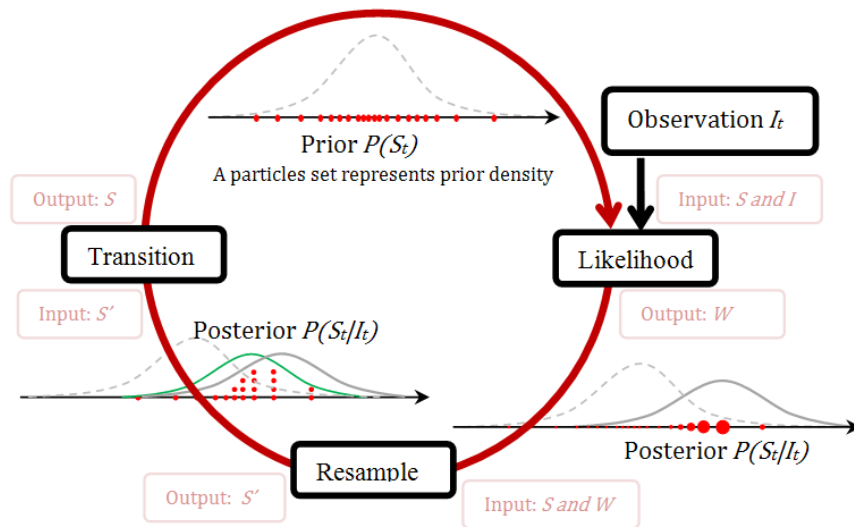



FIGURE 3.1: SIR particle filter consists of 3 functions likelihood, re-sample and transition.

Figure 3.1 shows the process of the SIR particle filter. At time t a state vector is denoted by S_t . A prior density function $P(S_t)$ and the observation Image I_t are fed to the likelihood function to produce a posterior density $P(S_t|I_t)$. The posterior is re-sampled and transformed by the transition function to generate the prior density of the next generation $P(S_{t+1})$.

$S =$

n	1	2	...	n_{max}
posx			...	
posy			...	
velx			...	
vely			...	



 $S_{n=2}$

FIGURE 3.2: Data structure of the state S .

Data-structure The particle filter uses the distribution of particles to represent the probability density function. So the subject state is expressed by n_{max} particles. Consider data-structure of the state vectors and weights, the n^{th} particle is a column vector of the state $S_{n,t} \in \mathbb{R}^{m_{max}}$, where each value in a dimension stores a physical attribute of the subject. Note that the time index t will be dropped because we are considering only single iteration.

Likelihood function In Algorithm 3.1, every single particle S_n is tested by the similarity measure. A value of similarity between a particle and observation is saved in a particle weight w_n . A combination between a set of particles and a set of weights builds the mass distribution in state space, which expresses the posterior density.

ALGORITHM 3.1: Likelihood function

- 1 Input: An observation image I and a particles set S
- 2 Output: A set of output weight w
- 3 For $n = 1$ to n_{max}
- 4 Generate a synthesis image I^s from a particle S_n
- 5 Calculate a similarity value between the synthesis I^s and the observation I^o and save the similarity to w_n
- 6 End

Resampling The re-sampling generates a new set of S by the inversion sampling method as described in Section 2.5.1.3. The cumulative sum of weight (**csw**) is calculated. Note that the sum of all weights must be normalised to 1. The **csw** is a function of n and it is monotonically increasing with respect to n . According to the inversion sampling method, a dummy set of numbers $\{u_i; u_i = \frac{i}{n_{max}}\}$ is created, which are uniformly distributed. The range of **csw**(n) and u_i is from 0 to 1. So u_i can be mapped to **csw**(n). A new sample index n' is sampled by inversion sampling.

$$u = \mathbf{csw}(n) \quad (3.1)$$

$$n'_i = \mathbf{csw}^{-1}(u_i) \quad (3.2)$$

\mathbf{csw}^{-1} is not a bijection function so that a single u_i could generate many n'_i depending on slope of **csw**(n). The more weight w_n the more the slope of **csw**(n) and the more the number of children. This method is called resampling [112]. The re-sample produces a new set of S . The child particles $S'_{n'}$ are at the same position as their mother particles S_n and the number of children is proportional to the weight w_n . Algorithm 3.2 shows detail of the re-sampling method.

ALGORITHM 3.2: Re-sample function

```

1  Input:  $S$  and  $w$ 
2  Output:  $S'$ 
3  Allocate memory for csw
4  Compute sum of weight  $\mathbf{sumw} \leftarrow \sum_{n=1}^{n_{max}} w_n$ 
5  For every  $n$ 
6    Normalise weight by  $w_n \leftarrow w_n / \mathbf{sumw}$ 
7  End
8   $\mathbf{csw}_1 \leftarrow w_1$ 
9  For  $n = 2$  to  $n_{max}$ 
10    $\mathbf{csw}_n \leftarrow \mathbf{csw}_{n-1} + w_n$ 
11 End
12  $i \leftarrow 0$ 
13 For  $n = 1$  to  $n_{max}$ 
14    $u \leftarrow (n - 0.5) / n_{max}$ 
15   While  $u > \mathbf{csw}_n$ 
16      $i \leftarrow i + 1$ 
17   End
18    $S'_n = S_i$ 
19 End

```

Transition function Then the child particles are transformed by the transition function as summarised in Algorithm 3.3. In the single target tracking problem, the state vector consists of the position on the ground floor plane and the velocity. The transition is a simple equation of motion.

$$S_{n,t} = \begin{bmatrix} \text{posx} \\ \text{posy} \\ \text{velx} \\ \text{vely} \end{bmatrix} \quad (3.3)$$

$$S_{n,t+1} = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} S_{n,t} + \begin{bmatrix} \sigma_{px} \mathcal{W}_{px} \\ \sigma_{py} \mathcal{W}_{py} \\ \sigma_{vx} \mathcal{W}_{vx} \\ \sigma_{vy} \mathcal{W}_{vy} \end{bmatrix} \quad (3.4)$$

The noise of velocity and position are modeled as the 2D *Wiener* processes [159], $\{\mathcal{W}_{px}, \mathcal{W}_{py}\}$ and $\{\mathcal{W}_{vx}, \mathcal{W}_{vy}\}$ are joint 2D Wiener processes of position and velocity, respectively. And variances ($\sigma_{px}, \sigma_{py}, \sigma_{vx}$ and σ_{vy}) control the distributions of the noise.

ALGORITHM 3.3: Transition function

<pre> 1 <u>Input</u>: S'_n 2 <u>Output</u>: S_n 3 For every n 4 Apply deterministic transformation $S_n \leftarrow AS'_{n,t}$ 5 Add noise to state vector $S_n \leftarrow S_n + \sigma \mathcal{W}$ 6 End </pre>

3.1.2 Basic similarity measure

In this section we will discuss a primitive version of the similarity measure for single target tracking. In order to compare the observation and a synthetic image, some useful pixels were extracted from the original image. Background segmentation was exploited to separate moving people from a static scene. The adaptive mixture of Gaussian (MOG) model [83] was used for background learning. Each pixel p of observation I_t was processed by a MOG filter. After 100 frames ($t = 1$ to 100) the MOG could estimate means $\mu_{c,p}$

and variances $\sigma_{c,p}^2$ of three colour channels $c = \{1, 2, 3\}$ (red, green and blue) of the pixel p . Then we used Chi-square test to separate the static background pixels from the dynamic foreground pixels.

$$Q_{t,p} = \sum_{c=1}^3 \left(\frac{I_{t,c,p} - \mu_{c,p}}{\sigma_{c,p}} \right)^2 \quad (3.5)$$

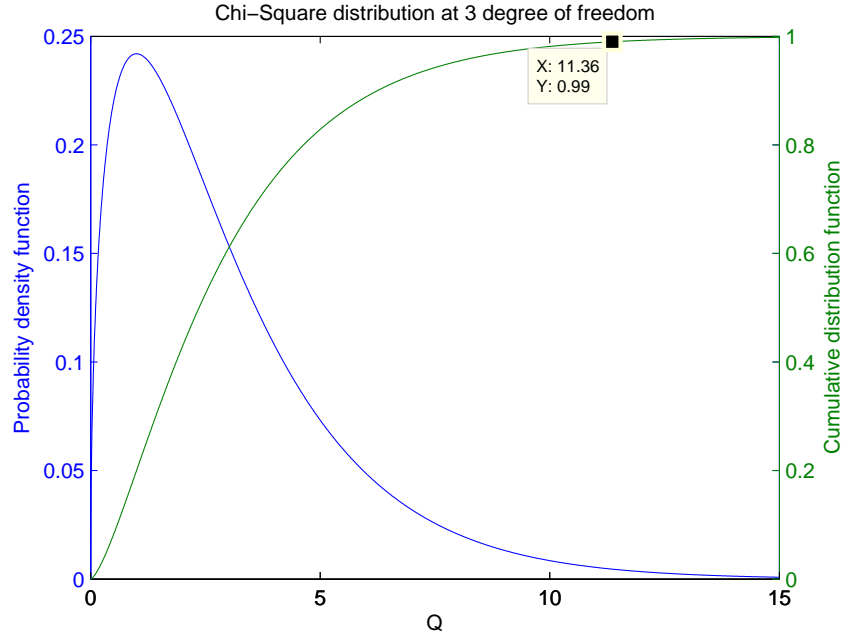


FIGURE 3.3: Chi-square distribution (blue) and its cumulative function (green)

Considering a particular pixel, Q is distributed as a Chi-square distribution. We can set a threshold based on probability. Figure 3.3 shows that 99% of observations of $Q_{t,p}$ are less than 11.36. This means 99% of measurements that come from the static background are in the range $0 < Q < 11.36$. Normally, moving objects have a distinctive colour compared to the background. So, we can separate the static background from moving objects by the Chi-square threshold.

$$I_{t,p}^f = \begin{cases} 0 & , Q_{t,p} < 11.36 \\ 1 & , 11.36 \leq Q_{t,p} \end{cases} \quad (3.6)$$

A foreground image I^f was generated by the segmentation method as described above. Then a similarity measure is computed using the Jaccard index between I^f and a synthetic image I^s .

In order to produce the synthetic image I^s , a rectangular plane was generated by vertical cross-section of a cylinder as shown by Figure 3.4. Note that the height and width of the cylinder were fixed. The normal vector \mathbf{n} of the rectangular surface does not has vertical components and it always points to the camera. This rectangular surface is generated by mapping 4 corner points of a cross-section model in real-world coordinates into the image plane as shown by Figure 3.5. In this experiment, state variables are the position of the cylinder centre on the ground plane and the velocity on the ground plane.

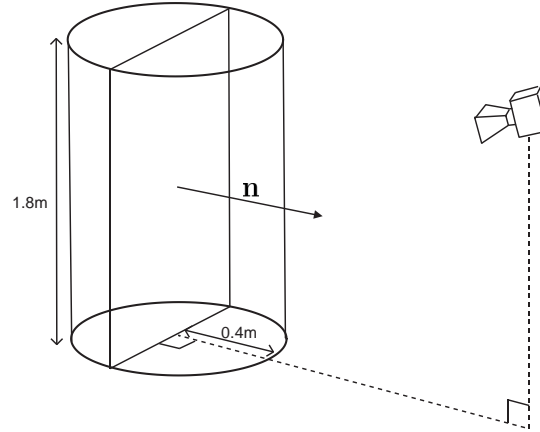


FIGURE 3.4: Cross-section of cylindrical model.

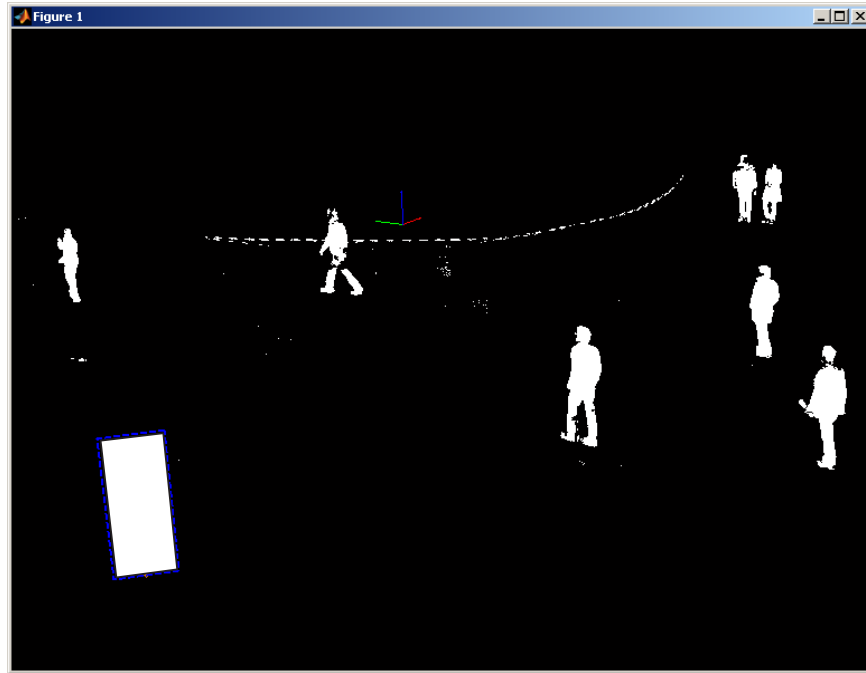


FIGURE 3.5: A rectangle on bottom-left is an example projection of the cylindrical cross-section model.

The corners of the rectangular model in the world coordinate system were projected to the image plane by the camera projection matrix \mathbf{P} . The projected template \mathbf{M} consists of 4 points (corners), where each point is expressed by a column vector in Equation (3.7). The r and h are radius and height of the cylindrical model.

$$\mathbf{M} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -r & -r & r & r \\ 0 & h & 0 & h \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad (3.7)$$

Before the projection, the template \mathbf{M} is transformed by rotation matrix to make the surface always faces to the camera. The rotation matrix is generated from the unit normal vector ($\mathbf{n} = [n_x, n_y]^T$), which is determined by the location of camera and the state position vector, $[\text{posx}, \text{posy}]^T$, as shows in Figure 3.4. The position vector is also used as a translation vector to shift the template from the origin to the position. Note that \mathbf{m} is a homogeneous coordinate system so the third element must be normalised to 1. Therefore, the 4 points of \mathbf{m} on image plane can be computed from Equation (3.8). Note that the projection of the model is a quadrilateral consisting of 4 corners, which need to be 90 degrees.

$$\mathbf{m} = \mathbf{P} \left(\begin{bmatrix} n_x & -n_y & 0 & \text{posx} \\ n_y & n_x & 0 & \text{posy} \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{M} \right) \quad (3.8)$$

Once the 4 points are projected, the ratio of foreground pixels in the quadrilateral over the total area was calculated. The ratio is the similarity between I^f and I^s , where I^s is a synthetic image of the quadrilateral and I^f is the foreground image. I^f is generated from the observation and the background model, whereas the rectangle image I^s is generated from the particle state.

For the n^{th} particle the likelihood is calculated form the ratio and stored in the weight w_n as Equation (3.9), where the quadrilateral image is generated by a particle S_n . The quadrilateral image is denoted by I_n^s . The sensitivity α makes similarity function sharper.

$$w_n = \left(\frac{|I^f \cap I_n^s|}{|I_n^s|} \right)^\alpha \quad (3.9)$$

The particle filter used 512 particles so two thousands corners were produced in each single observation image. The computation cost is relatively high when compare to maximum-likelihood method.

3.1.3 Experiment and result

In the experiment of single target tracking, the SIR particle filter was implemented with 512 particles and sensitivity $\alpha = 5$. The program was implemented in MATLAB and tested on the PETS09 dataset [160].

PETS09 provides both intrinsic and extrinsic camera parameters in xml files. However, the dataset gave the orientation in terms of Euler angle $[R_x \ R_y \ R_z]$ (a 3-by-1 vector), so it was converted to a 3-by-3 rotation matrix. The conversion method can be found in the xyz (pitch-roll-yaw) Euler Angles convention method[161] as shows in Equation (3.10).

$$R = \begin{bmatrix} \cos R_y \cos R_z & \cos R_z \sin R_x \sin R_y - \cos R_x \sin R_z & \sin R_x \sin R_z + \cos R_x \cos R_z \sin R_y \\ \cos R_y \sin R_z & \sin R_x \sin R_y \sin R_z + \cos R_x \cos R_z & \cos R_x \sin R_y \sin R_z - \cos R_z \sin R_x \\ -\sin R_y & \cos R_y \sin R_x & \cos R_x \cos R_y \end{bmatrix} \quad (3.10)$$

The program was tested on a work station computer with a 2.4GHz CPU. Its running speed was about 0.2 frames per second (fps). The system required an initialization position by manually selecting a subject.

Figure 3.6 is an example result from many repetitive tests. The estimation was very close to ground truth in the early period of tracking. Until around frame 300 the subject was occluded by another person and the static object led to drifting of the estimation. The tracker sought the subject and oscillated around the ground truth towards the end period of tracking. The drifting was about 2m away from the ground truth but it appeared as a few pixels (20 pixels) in the image plane. When the angle between the camera direction and the ground plane is small it amplifies the error from a few pixels in the vertical direction on the image to a huge distance on the ground plane. If the camera

angle increases the error distance should decrease. Another solution is to design a new similarity measure, which produces higher precision measure in the vertical direction on the image plane.

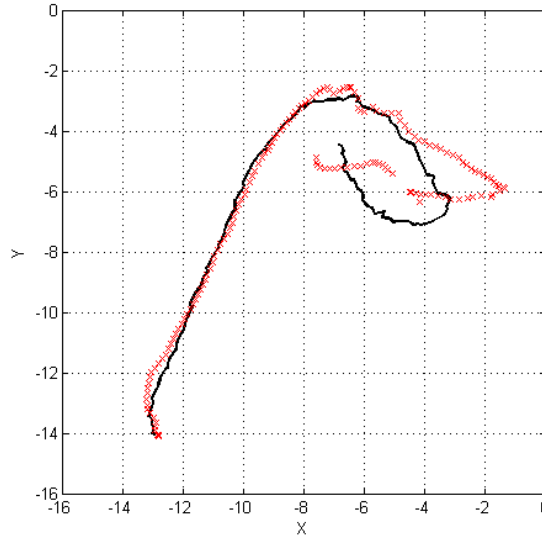


FIGURE 3.6: Black solid line is ground truth and red markers are the estimated trajectory in meter.

Figure 3.7 shows a capture screen of the program during execution. The captured screen at frame 350 shows a distraction from the person on the right side resulting in a loss of target in the next frame. The result shows many problems.

- *Firstly*, the drifting effect due to the imprecise similarity measure in the vertical direction in the image plane leads to a large error in the radial direction to the camera. When a subject is further away from the camera the angle between the ground plane and a direction from the camera to the target is small and caused larger error. It is similar to find an intersection between 2 lines where the angle between them is really small and the intersection point is hard to detect perfectly.
- *Secondly*, the similarity measure computation is slow at 0.2 fps so it is far from a practical real-time application. The number of particles can be reduced but this will affect the precision. So reducing the number of particles is not a good idea. The similarity measurement must be improved in term of speed and precision.

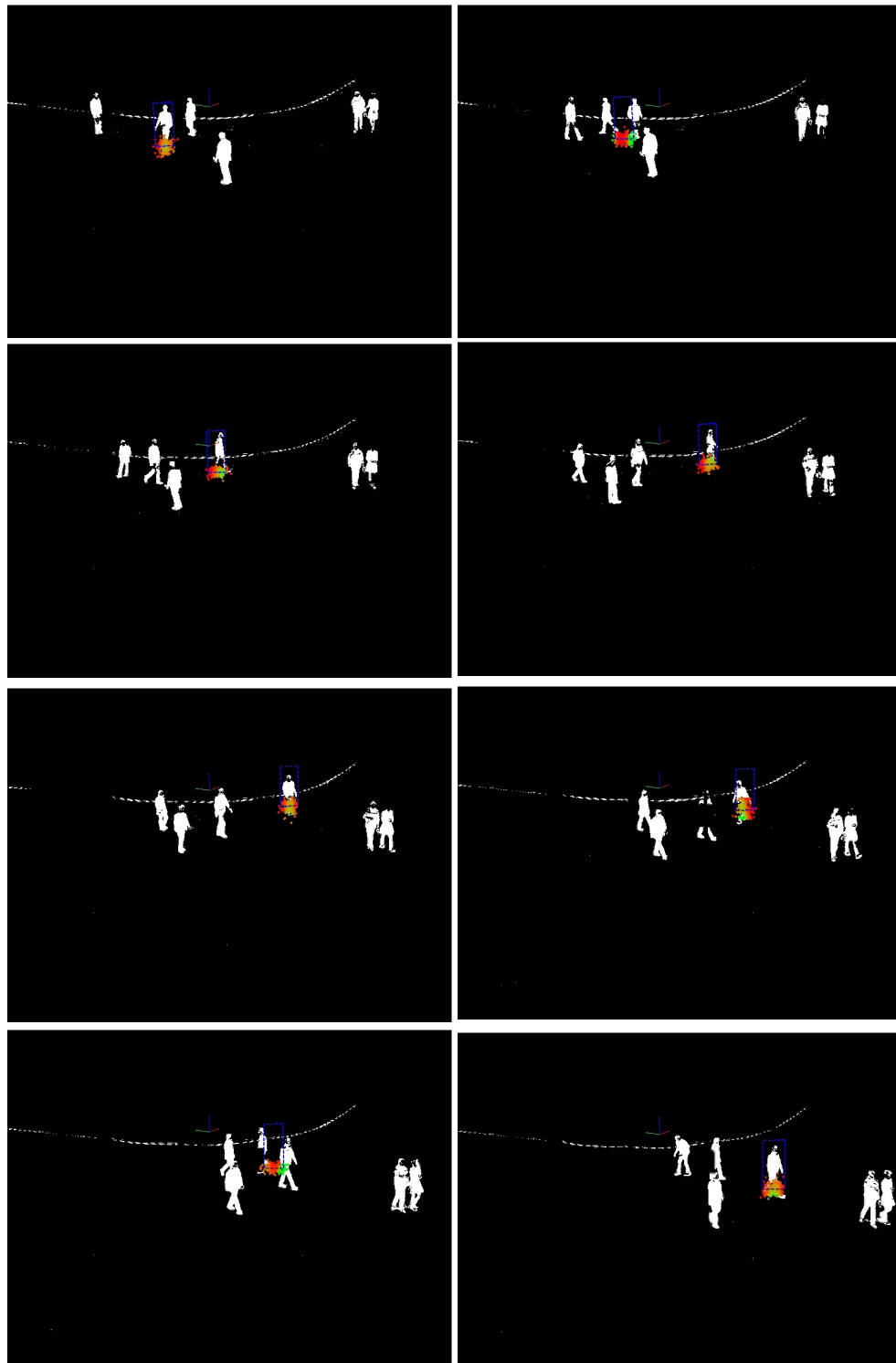


FIGURE 3.7: From top-left to bottom-right, tracking results of frame 290, 300, 310, 320, 330, 340, 350 and 360. Particles on ground floor show estimated position where green is high weight. The tracker was distracted in the last two frames.

- *Thirdly*, the system is not fully automatic as it requires human to select a subject

position to initialise the tracker. Applying the object detection method can full-fill this requirement. This means that detection and tracking function must run concurrently. So, more computation power is required to achieve a higher frame rate.

- *Finally*, distractions from other moving objects are inevitable. In order to solve this problem, the tracking system must know all subject positions in the scene. Once we know the positions of all subjects, the overlapping between their silhouettes can be predicted and handled effectively.

3.2 Fast Ellipsoid projection

In previous sections, the simple tracking method was implemented and tested. The problems, precision, distraction and speed, have been identified. The problems are caused by a too simple template model. In this section we will introduce a new likelihood function based on an ellipsoid projection which can prevent those problems that occurred in the simple tracking. The new model can measures the similarity more precisely to suppress the distraction problem.

3.2.1 Ellipsoid projection

A surface of a human body is approximated as an ellipsoid model, which is a stretched sphere in a particular direction. An equation of a unit sphere is shown in Equation (3.11), where a point $\mathbf{q} = [q_x \ q_y \ q_z]^T$ is on the unit sphere surface.

$$q_x^2 + q_y^2 + q_z^2 = 1 \quad (3.11)$$

We can stretch each dimension by applying a scaling factor as in Equation (3.12) and this is known as a ellipsoid equation. The scaling factor a_x, a_y and a_z are radii for the x, y and z directions, respectively.

$$\left(\frac{q_x}{a_x}\right)^2 + \left(\frac{q_y}{a_y}\right)^2 + \left(\frac{q_z}{a_z}\right)^2 = 1 \quad (3.12)$$

We can express the ellipsoid in a matrix form as in Equation (3.13).

$$\mathbf{q}^T \mathbf{A} \mathbf{q} = 1 \quad (3.13)$$

The scaling matrix \mathbf{A} consists of the inverse of the radii as shown in Equation (3.14)

$$\mathbf{A} = \begin{bmatrix} \frac{1}{a_x^2} & 0 & 0 \\ 0 & \frac{1}{a_y^2} & 0 \\ 0 & 0 & \frac{1}{a_z^2} \end{bmatrix} \quad (3.14)$$

The ellipsoid can be translated from an origin point to a particular position \mathbf{s} as shown in Equation (3.15) and this equation is used as the human body template.

$$0 = 1 - (\mathbf{q} - \mathbf{s})^T \mathbf{A} (\mathbf{q} - \mathbf{s}) \quad (3.15)$$

In order to project the ellipsoid template to the image plane, the ray tracing technique of ellipsoid projection [162] has been extended. The ray tracing method is originally from computer graphics and used to generate an artificial image from a predefined 3D environment. By setting a camera at a particular location in 3D space, a ray that passes through the camera origin and a pixel can be determined. In Section 2.6 the camera model and calibration method have been reviewed. The origin point of a camera coordinate system is a point that all light rays must pass through, also known as *an eye-point*. The ray tracing projection draws a ray (a line in 3D) from the eye-point passing through a pixel and continuing further until the line intersects some surface. Once the ray meets a surface, the pixel colour and intensity are determined by the surface property. For example, if the ray meets a red cube in 3D space the pixel is assigned to be red. So, we need to know how to compute the eye-point and the direction.

In the camera model the extrinsic parameters explain the orientation and translation of the camera. The camera coordinate system can be expressed in terms of a world coordinate system as in Equation (3.16).

$$\mathbf{q}_{camera} = \mathbf{R} \mathbf{q}_{world} + \mathbf{t} \quad (3.16)$$

$$\mathbf{q}_{world} = \mathbf{R}^T (\mathbf{q}_{camera} - \mathbf{t}) \quad (3.17)$$

From the camera model we know that the eye-point is an origin point in the camera coordinate system, $\mathbf{q}_{camera} = \mathbf{0}$. So, we can compute the eye-point, which is denoted by \mathbf{e}_{world} , from Equation (3.18). The \mathbf{e}_{world} is position of the origin of the camera model.

$$\mathbf{e}_{world} = -\mathbf{R}^T \mathbf{t} \quad (3.18)$$

Next the intrinsic matrix \mathbf{K} transforms a coordinate \mathbf{q}_{camera} on the image plane to a pixel coordinate $\mathbf{p}_{image} = [p_x \ p_y \ 1]^T$.

$$\mathbf{p}_{image} = \mathbf{K} \mathbf{q}_{camera} \quad (3.19)$$

$$\mathbf{q}_{camera} = \mathbf{K}^{-1} \mathbf{p}_{image} \quad (3.20)$$

Substitute Equation (3.20) into 3.17.

$$\mathbf{q}_{world} = \mathbf{R}^T (\mathbf{K}^{-1} \mathbf{p}_{image} - \mathbf{t}) \quad (3.21)$$

Then the ray direction \mathbf{d} is computed from the coordinate \mathbf{q}_{world} and the eye-point \mathbf{e}_{world} .

$$\mathbf{d} = \mathbf{q}_{world} - \mathbf{e}_{world} \quad (3.22)$$

$$\mathbf{d} = \mathbf{R}^T \mathbf{K}^{-1} \mathbf{p}_{image} \quad (3.23)$$

In summary, we can compute the eye-point and the direction from Equation (3.18) and Equation (3.23), respectively.

A line in 3D space is a representation of light ray which is constructed from the eye-point and the ray direction. The ray is a collection of infinite points that meets the condition in Equation (3.24), where \mathbf{d} is a unit vector of the ray direction and τ is the distance from the eye-point to the ellipsoid surface. Note that we have to drop the subscript $(\)_{world}$ to simplify the equation and we are now considering every vector in the 3D world coordinate.

$$\mathbf{q}(\tau) = \mathbf{e} + \tau \mathbf{d} \quad (3.24)$$

The equation of ray, Equation (3.24), is inserted in the ellipsoid, Equation (3.15) and the output is Equation (3.25).

$$0 = 1 - \tau^2 \mathbf{d}^T \mathbf{A} \mathbf{d} - 2\tau \mathbf{d}^T \mathbf{A}(\mathbf{e} - \mathbf{s}) - (\mathbf{e} - \mathbf{s})^T \mathbf{A}(\mathbf{e} - \mathbf{s}) \quad (3.25)$$

$$0 = \alpha \tau^2 + \beta \tau + \gamma \quad (3.26)$$

Where

$$\alpha = \mathbf{d}^T \mathbf{A} \mathbf{d} \quad (3.27)$$

$$\beta = 2\mathbf{d}^T \mathbf{A}(\mathbf{e} - \mathbf{s}) \quad (3.28)$$

$$\gamma = (\mathbf{e} - \mathbf{s})^T \mathbf{A}(\mathbf{e} - \mathbf{s}). \quad (3.29)$$

From Equation (3.26), there are three cases that determine the root and each case corresponds to a particular intersection situation as in the list below.

1. The roots τ are two different real numbers which means the ray intersects the ellipsoid surface twice.
2. The root τ are two different imaginary number which means the ray never intersects the ellipsoid.
3. The root is a single real value when the ray just touches the ellipsoid surface.

We are considering the third case when the ray intersects the ellipsoid once which implies that the ray touches the ellipsoid at a boundary. We can construct a projection of the ellipsoid silhouette on the image plane by rays that meet the third condition. A general form of the root of a quadratic equation is

$$\tau = \frac{-\beta \pm \sqrt{\beta^2 - 4\alpha\gamma}}{2\alpha} \quad (3.30)$$

The root is a single real value when the discriminant $\beta^2 - 4\alpha\gamma$ is zero, where $\mathbf{B} = \mathbf{A}(\mathbf{e} - \mathbf{s})$ and \mathbf{I} is the identity matrix.

$$\beta^2 - 4\alpha\gamma = 4[\mathbf{d}^T \mathbf{A}(\mathbf{e} - \mathbf{s})]^2 - 4[\mathbf{d}^T \mathbf{A} \mathbf{d}][(\mathbf{e} - \mathbf{s})^T \mathbf{A}(\mathbf{e} - \mathbf{s}) - 1] \quad (3.31)$$

$$= 4\mathbf{d}^T [\mathbf{B}\mathbf{B}^T - \mathbf{B}^T \mathbf{B} \mathbf{I} + \mathbf{A}] \mathbf{d} \quad (3.32)$$

Next Equation (3.23) is substituted into Equation (3.32).

$$\beta^2 - 4\alpha\gamma = 4\mathbf{p}^T \mathbf{K}^{-T} \mathbf{R}^T [\mathbf{B}\mathbf{B}^T - \mathbf{B}^T \mathbf{B} \mathbf{I} + \mathbf{A}] \mathbf{R}^T \mathbf{K}^{-1} \mathbf{p} \quad (3.33)$$

$$= 4\mathbf{p}^T \mathbf{E} \mathbf{p} \quad (3.34)$$

The conic matrix \mathbf{E} is symmetric. It transforms a pixel coordinate to a scalar ψ , which can identify whether the ray from \mathbf{p} intersects the ellipsoid or not. When ψ is positive, the τ has two roots and the ray intersects the ellipsoid surface twice. The ray never intersects the ellipsoid if ψ is negative value. In the special case when the ray intersects once, the function $\psi = 0$ defines an elliptical bounding box of the projected ellipsoid.

$$\psi = \frac{\beta^2 - 4\alpha\gamma}{4} \quad (3.35)$$

$$\psi = \mathbf{p}^T \mathbf{E} \mathbf{p} \quad (3.36)$$

The scalar function ψ is a function of the pixel coordinate (p_u, p_v) . The conic function in the matrix form has been transformed to a scalar form to reduce the complexity of computation. In the likelihood computation, the ψ value on each considered pixel must be calculated and the scalar form improves speed a lot. If the pixel-wise computation is complex the global computation is much more complex, so, we have to reduce complexity from the pixel level. The Equation (3.36) needs 12 multiplications and 8 additions, whilst, Equation (3.37) has 6 multiplications and 8 additions. For huge number of pixels the different is amplified and results in a speed gain of several milliseconds.

$$\psi = 1 - A(p_u - c_u)^2 - B(p_u - c_u)(p_v - c_v) - C(p_v - c_v)^2 \quad (3.37)$$

The scalar coefficients, A, B and C , characterise size and orientation of the ellipse. The centre point (c_u, c_v) can be computed from the derivative of ψ in Equation (3.36) with respect to p_u and p_v because the ψ is a convex function and has a peak at the centre. The centre position is computed from the conic matrix \mathbf{E} .

$$c_x = \frac{E_{12}E_{23} - E_{13}E_{22}}{E_{11}E_{22} - E_{12}^2} \quad (3.38)$$

$$c_y = \frac{E_{12}E_{13} - E_{11}E_{23}}{E_{11}E_{22} - E_{12}^2} \quad (3.39)$$

The major and minor radii (a and b) are calculated from Equation (3.41) and Equation (3.42).

$$\mathbf{E}^* = \begin{bmatrix} E_{11} & E_{12} \\ E_{12} & E_{22} \end{bmatrix} \quad (3.40)$$

$$\frac{1}{a^2} = -\lambda_1 \frac{\det \mathbf{E}^*}{\det \mathbf{E}} \quad (3.41)$$

$$\frac{1}{b^2} = -\lambda_2 \frac{\det \mathbf{E}^*}{\det \mathbf{E}} \quad (3.42)$$

Where λ_i and $\mathbf{V}_i = [V_{1i} \ V_{2i}]^T$ are an eigenvalue and a corresponding column eigenvector of the \mathbf{E}^* . The coefficients in Equation (3.37) are computed from the eigenvectors and radii. So we can compute all five parameters in Equation (3.37).

$$A = \frac{V_{11}^2}{a^2} + \frac{V_{11}^2}{b^2} \quad (3.43)$$

$$B = 2 \left(\frac{V_{11}V_{21}}{a^2} + \frac{V_{12}V_{22}}{b^2} \right) \quad (3.44)$$

$$C = \frac{V_{21}^2}{a^2} + \frac{V_{22}^2}{b^2} \quad (3.45)$$

To summarise this section, the ellipsoid model represents a human body surface in a 3D world and a light ray is computed from a direction vector \mathbf{d} and the eye-point \mathbf{e} , which are derived from a camera model. The ray is expressed in term of τ , which is the distance from the eye-point to the ellipsoid surface. τ is root of the quadratic function and the special case is when the ray touches the ellipsoid and that defines the boundary of the ellipsoid projection on image. We constructed the boundary contour equation ψ which is a general form of ellipse equation. The ellipse equation can be simplified to a scalar function and it has lower complexity compared to the vertex form.

In next section, we will described texture mapping, which was used as a feature in tracking and allowed the system to perform recognition.

3.2.2 Texture mapping

The ellipsoid surface is a 2D closed surface in 3D space. The colour distribution on the 2D surface is called *a texture image* as in computer graphics. There are so many method to map a point in a coordinate system to another system. The ellipsoid surface

in not flat and a 2D Euclidean coordinate system is unable to represent position on the ellipsoid surface. A coordinate of an ellipsoid surface is mapped to a cylindrical coordinate system, which makes the coordinate on the surface manageable.

We used a cylindrical coordinate system to map a point on the ellipsoid surface to the texture image because the cylindrical mapping makes a uniform distribution of points on an ellipsoid surface [163]. A benefit of using a uniform distribution mapping is that each pixel on a texture image represents an equal surface area on the ellipsoid surface. All pixels of the texture image are equally important.

To calculate a point on an ellipsoid surface, which intersects a ray, we have to reconsider Equation (3.26). The distance τ must be calculated and then intersection point in 3D Euclidean space will be mapped to a 2D cylindrical coordinate. In order to computer surface to surface mapping, we scale the ellipsoid to a unit sphere. It makes calculation much easier.

The scaling factor is a square root of \mathbf{A} , where a_x, a_y and a_z are radii of the ellipsoid.

$$\mathbf{A}^{\frac{1}{2}} = \begin{bmatrix} \frac{1}{a_x} & 0 & 0 \\ 0 & \frac{1}{a_y} & 0 \\ 0 & 0 & \frac{1}{a_z} \end{bmatrix} \quad (3.46)$$

From Equation (3.26) we can express this in a scaling space $\hat{\mathbf{d}} = \mathbf{A}^{\frac{1}{2}} \mathbf{d}$ and $\hat{\mathbf{e}}_s = \mathbf{A}^{\frac{1}{2}} (\mathbf{e} - \mathbf{s})$.

$$0 = -\hat{\mathbf{d}}^T \hat{\mathbf{d}} \hat{\tau}^2 - 2\hat{\mathbf{d}}^T \hat{\mathbf{e}}_s \hat{\tau} + (1 - \hat{\mathbf{e}}_s^T \hat{\mathbf{e}}_s) \quad (3.47)$$

Before the $\hat{\tau}$ is solved the direction vector $\hat{\mathbf{d}}$ must normalised to a unit, let $\bar{\mathbf{d}} = \frac{\hat{\mathbf{d}}}{|\hat{\mathbf{d}}|}$. Now we can compute the distance $\hat{\tau}$ and the intersection coordinate $\mathbf{u} = [u_1 \ u_2 \ u_3]^T$ on an ellipsoid surface from Equation (3.48) and 3.49.

$$\hat{\tau} = -\bar{\mathbf{d}}^T \hat{\mathbf{e}}_s - \sqrt{\psi} \quad (3.48)$$

$$\mathbf{u} = \hat{\mathbf{e}}_s + \hat{\tau} \bar{\mathbf{d}} \quad (3.49)$$

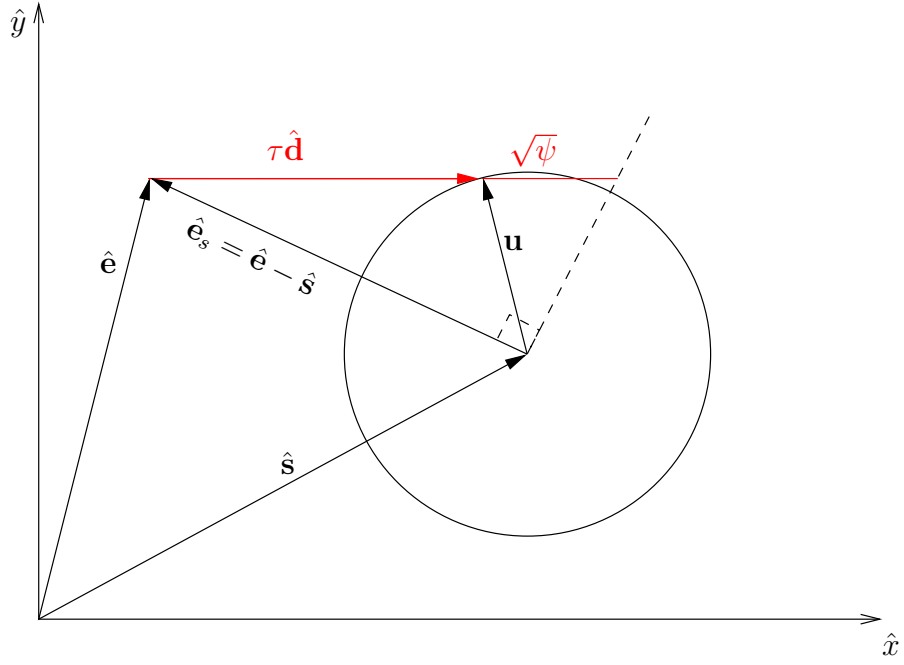


FIGURE 3.8: A top-view of the projection system, eye-point $\hat{\mathbf{e}}$, position of ellipsoid $\hat{\mathbf{e}}$ and the direction vector $\hat{\mathbf{d}}$.

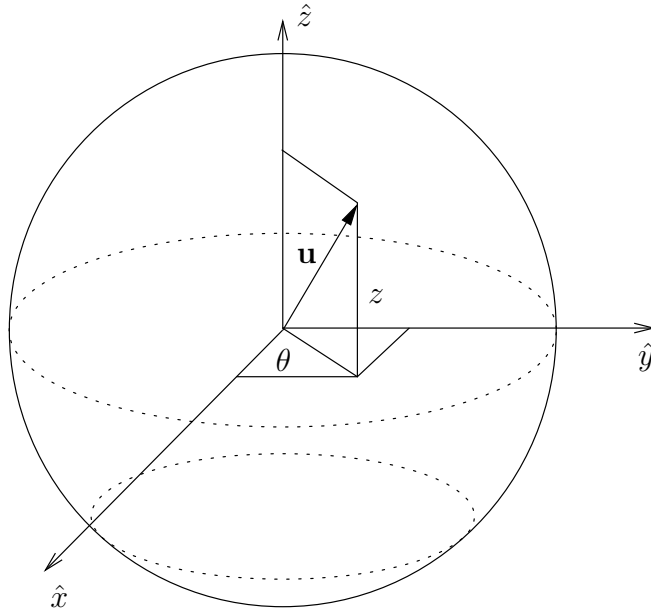


FIGURE 3.9: Transforming Euclidean coordinate to Cylindrical coordinate.

The unit vector $\mathbf{u} = [u_1 \ u_2 \ u_3]^T$ points from the centre of the unit sphere to its surface as shown in Figure 3.9.

$$z = u_3 \quad (3.50)$$

$$\theta = \tan 2(u_2, u_1) - \theta_o \quad (3.51)$$

We can transform the coordinate system from Euclidean to a cylindrical coordinate as in Equation (3.50) and 3.51. The function $\text{atan2}(x, y)$ is a arctangent function and the θ_o is a facing angle of the subject, which is approximated by the velocity of the subject.

3.2.3 Summary

In this section, we model a human body as a 3D ellipsoid. The 3D ellipsoid is transformed to a 2D ellipse and the transformation allows us to calculate a contour function ψ in parametric form. The parametric ellipse contour function will accelerate silhouette likelihood computation. We use a scaling transformation to make an ellipsoid to be a sphere in order to calculate the distance τ . When τ is computed we can find the intersection coordinate on the unit sphere. Then the intersection coordinate on the unit sphere surface in Euclidean coordinate system is transformed to a cylindrical coordinate system to make uniform texture mapping. Finally, we can map ellipsoid surface to the texture image in the (z, θ) coordinate system.

3.3 Ellipsoid Likelihood

In this section, we will use the parametric ellipse contour function in order to compute the likelihood. The ellipse Equation (3.37) is an elliptical boundary on an image plane. The elliptical boundary is used in the likelihood calculation by computing the fill-in ratio as in simple tracking in Section 3.1.2. The likelihood also includes other features such distraction suppression and a texture similarity measure.

The computation of the parameters in an ellipse function is more complex than in the previous rectangular model but the ray-tracing calculation for quadrilateral surface is more complex than the ellipse function. When considering many pixels in a boundary, the ellipse function will be faster. To show the complexity of ray tracing in the case of a rectangular plan model constructed from 4 vertexes. : in [164] (page 482-491), the rectangular template is constructed from 4 points (4 vertexes or 2 triangles).

Vertex base ray tracing: The procedure to compute ray tracing for a vertex model is below.

- (1) The triangles are rotated and translated to a desired position.

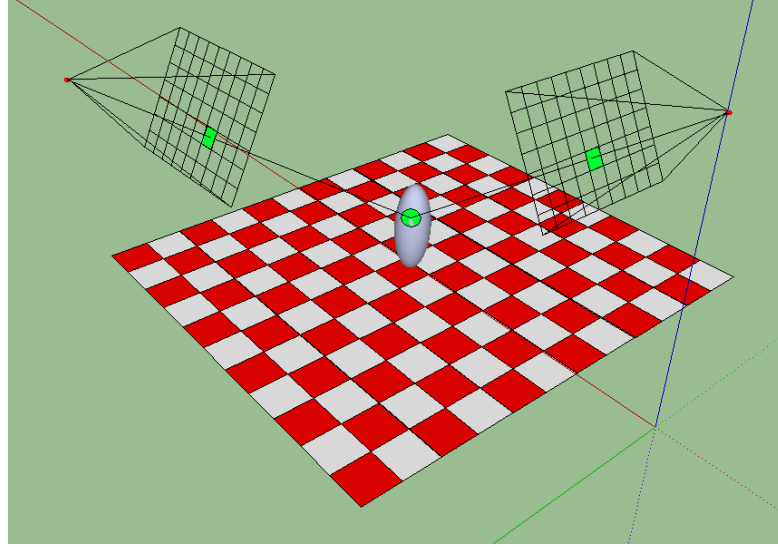


FIGURE 3.10: Ray tracing of an ellipsoid in 2 cameras system.

- (2) The distance from the camera to the plane is computed.
- (3) The intersection point is expressed in term of vectors of edges of the triangle, also called *barycentric coordinates*.
- (4) The linear equation are solved to get the intersection point in barycentric coordinates.
- (5) If the barycentric coordinates are in the range between 0 to 1, so the ray intersects the triangle.

The template preparation in step (1) is computed for every frame and steps (2)-(5) are computed for every pixel.

Parametric ellipsoid ray tracing: in contrast, ray tracing for an ellipsoid model as below requires complex preparations in step (1) and (2), but requires a simple computation at the pixel level in steps (3) and (4).

- (1) The conic matrix \mathbf{E} is computed from Equation (3.33), which includes translating the ellipsoid to the desired location.
- (2) The ellipse parameters A, B, C, c_u, c_v are computed.
- (3) The contour level ψ is computed.
- (4) It is checked whether ψ is in the range from 0 to 1 or not, if so the ray intersects the ellipsoid.

Computation of ray tracing in a vertex base is expensive at the pixel level, whilst the ellipsoid model is cheaper. This makes the ellipsoid model more suitable for real-time tracking.

TABLE 3.1: Mathematical operation requirements for intersection calculation per object per p pixels

		Number of instruction calls					
		Add/Sub	Product	Division	Binary	Sqrt	Trigonometry
Vertex base v vertexes n planes	(1)	$9v$	$12v$	-	-	-	-
	(2)	$5np$	$6np$	$1np$	-	-	-
	(3)-(4)	$55np$	$36np$	$1np$	-	-	-
	(5)	-	-	-	$2np$	-	-
	Total	$9v + 60np$	$12v + 42np$	$2np$	$2np$	-	-
Parametric Ellipsoid	(1)	87	50	-	-	-	-
	(2)	21	39	4	-	1	6
	(3)	$7p$	$6p$	-	-	-	-
	(4)	-	-	-	$1p$	-	-
	Total	$108 + 7p$	$89 + 6p$	4	$1p$	1	6

Table 3.1 shows a comparison of operations required in a traditional vertex base [164], where the object is constructed from v vertexes and n planes. The number of pixels p to perform ray tracing is usually varying in range between a few thousands and millions of pixels. In the rectangular simple model, the model is constructed from 2 planes and 4 vertexes ($n = 2$ and $v = 4$). We assume that $p = 100,000$ pixels. Table 3.2 shows the total number of instruction calls in a unit of million. This estimation shows the difference of complexity, where the parametric ellipsoid ray tracing is about one order of magnitude faster than the vertex base.

TABLE 3.2: Comparison of the total number of calls in a unit of million, where $n=2$, $v=4$ and $p=100,000$

	Number of instruction calls $\times 10^6$					
	Add/Sub	Product	Division	Binary	Sqrt	Trigonometry
Vertex	12.0	8.4	0.4	0.4	0.0	0.0
Ellipsoid	0.7	0.6	0.1	0.0	0.0	0.0

3.3.1 Silhouette similarity

The Ellipsoid projection becomes more effective when considering a large number of pixels. This efficiency is based on the fact that the projection of an ellipsoid is an ellipse contour function as showed in Figure 3.11. Therefore, we can express an ellipsoid as a contour function that leads to a light complexity at the pixel level.

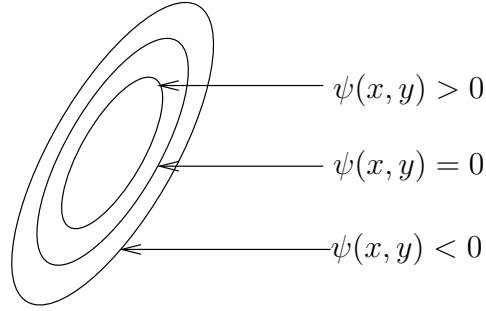


FIGURE 3.11: Ellipse Contour.

The second advantage of using the ellipsoid model is that we can scale the ellipse contour kernel by simply reducing the level of the contour ψ as shown in Figure 3.11. The boundary transformation by setting different contour values the method is also called *level-set method*. When the value of level-set reduces the elliptical bounding box is scaled up and this allow us to expand the considered region, to include a low level area that has low a probability to detect any foreground pixels. In this thesis, the kernel integration between a synthetic image and a foreground image are treated as a log-likelihood.

From the likelihood definition Equation (3.53), a template foreground pixel density function $g(p_i, S)$ is constructed from state S and an observation foreground pixel density $f(p)$. Note that the data structure I^f is a binary image, where a static background pixel at p_i is denoted by $I^f(p_i) = 0$ and an active foreground pixel on a moving objects has $I^f(p_i) = 1$.

$$\{I^f(p) > 1\} \sim f(p) \quad (3.52)$$

In Equation (3.53), p_i is a linear memory address of a pixel (i determined from coordinate u and v , $i = u + v \times u_{max}$). The logarithm function transforms multiplication to summation and Equation (3.55) is expressed in kernel integral form.

$$\mathcal{L}(I^f|S) = \prod_i g(p_i, S) \quad ; p_i \sim f(p) \quad (3.53)$$

$$\log [\mathcal{L}(I^f|S)] = \sum_i \log[g(p_i, S)] \quad (3.54)$$

$$\log [\mathcal{L}(I^f|S)] = \int f(p) \cdot \log[g(p, S)] dp \quad (3.55)$$

If we assume the template distribution of the foreground pixel is Normal distribution, the kernel will be a quadratic function as simple as the ellipse contour function. Let the

kernel be ψ and the foreground density function $f(p)$.

$$\log \left[\mathcal{L} \left(I^f | S \right) \right] = \int \left\{ I^f(p) > 1 \right\} . \psi(p, S) \, dp \quad (3.56)$$

Considering the image region from $-0.5 < \psi < 1$, a pixel in a positive contour region contributes a positive value to a log-likelihood summation. In contrast a pixel in a negative contour penalises the log-likelihood. Including the negative region into the likelihood computation improves accuracy of the likelihood function by penalising any misaligned pixel in the negative region. Figure 3.12 shows positive and negative contour regions which can prevent the drifting effect occurring in the previous simple tracking method.

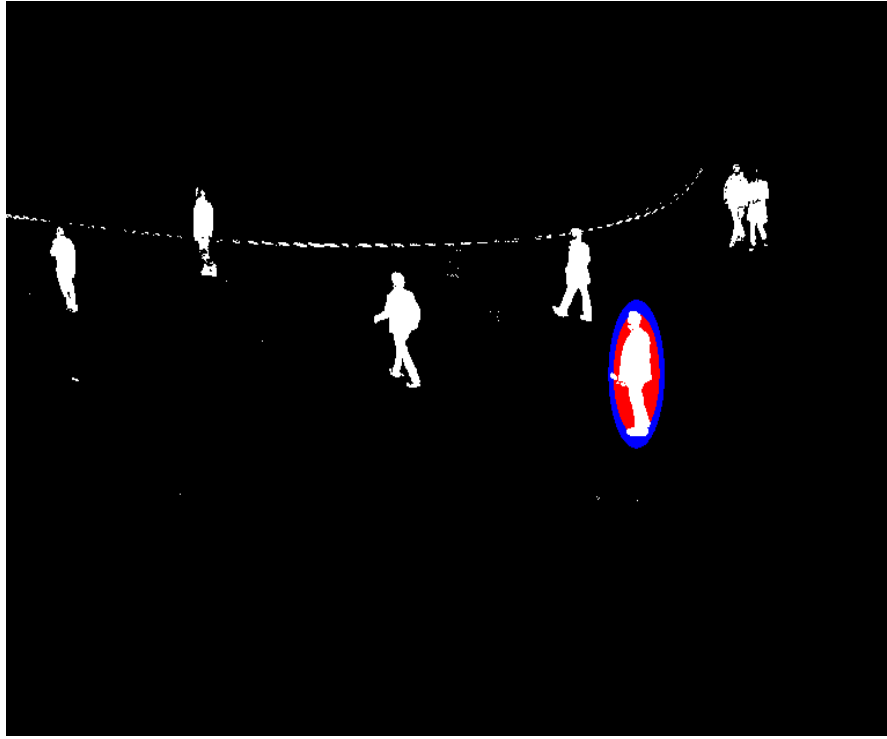


FIGURE 3.12: An ellipse kernel projected on a foreground image. Red area is positive contour, $0 \leq \psi$, and blue region is negative contour, $-0.5 < \psi < 0$.

However, from our experiment the distribution of active pixels in the elliptical boundary is not similar to Normal distribution. The active pixels spread uniformly in the boundary. So, we had to change from Normal distribution to Uniform distribution. This makes ψ in at positive region of the ellipse contour to be constant. The uniform

kernel, ψ^u in Equation (3.57), is used instead of the function ψ .

$$\psi^u = \begin{cases} 1 & ; 0 \leq \psi \\ -\epsilon & ; \psi > 0 \end{cases} \quad (3.57)$$

Another advantage of using an ellipsoid model to represent a human body is that it allows us to detect overlaps between different ellipses in an image very quickly by using the ellipse centre and the radii. This allows us to detect overlaps and model interaction between ellipses in order to reduce the distraction problem.

3.3.2 Ownership of a pixel

Imagine that there are two subjects being tracked and we are considering a pixel, which is at overlap region. The ray from the camera eye-point intersects both ellipses for all pixels in an overlap region. The pixels contribute the positive gains to both ellipses. But in fact the ownership of the pixel belongs to only one ellipse.

We have tested 2 ideas to solve the overlap interaction problem. In *the first method*, the closest ellipsoid from the camera is the owner of the pixel. This method is reasonable when you imagine that a ray from a camera intersects the closest object and the further objects are occluded. However, the method gives potential to the closest ellipsoid (nearest from the camera) to invade the further ellipsoids. The closest ellipsoid takes over the surrounding pixels that should belong to the further object. This leads to the drifting of the further object. On the image plane, further ellipsoids are pushed away from the closest due to this mechanism. The method normally perceives two people as single closer (bigger) person. To overcome this problem, we used the second method; ownership of the pixel is shared equally by all intersecting ellipsoids. If the ray from the pixel intersects m ellipsoids, the pixel will contribute $\frac{1}{m}$ in a log-likelihood summation. The normalising factor πab is area of the ellipse kernel.

$$\log [\mathcal{L}(I^f|S)] = \frac{1}{\pi ab} \sum_i \frac{1}{m} \psi^u(p_i, S) \quad ; p_i \sim f(p) \quad (3.58)$$

$$\log [\mathcal{L}(I^f|S)] = \frac{1}{\pi ab} \int \frac{1}{m} f(p) \cdot \psi^u(p, S) dp \quad (3.59)$$

The sharing ownership was better in the comparison experiment but it still not perfect. There was a small chance of distraction between subjects. However, this chance is relatively low when compared to the simple tracking, which has no distraction suppression.

3.3.3 Texture learning and texture likelihood

We also used texture in the similarity measure to prevent the distraction problem. In Section 3.2.2, we have described texture mapping between an ellipsoid surface and a texture image. We assumed that the texture is consistence even when the human body periodically changes. Because the motion of a body is periodic when we observe the subject for a long period we can estimate the mean and variance of the colour distribution of a particular surface area. Note that we assume that the facing angle of the ellipsoid is in the same direction of as the velocity vector. Therefore, the colour distribution of a pixel over a long period of observation can be estimated. We used the MOG method to estimate the colour distribution as same as in the background segmentation. The colour distribution estimation over time of a pixel in a texture image will be called *texture pixel colour estimation* (TPC). The TPC consists of the means and variances of all the colour channels. Immediately after initialising, the TPC is unknown, so the system must cumulatively learn the TPC from observations. At this stage, the object position estimation relies on the primary silhouette likelihood. After a certain period, the TPC is captured and it is ready to be added into the likelihood calculation.

The texture likelihood is computed from the summation of matching score between TPC and observation from ray tracing. In texture likelihood computation, we consider all foreground pixels in the elliptical boundary. Each pixel, which meets the condition, is used in the matching calculation. Any matching TPC is counted as positive and mismatching pixel is negative. We used the Chi-square test with 3 degrees of freedom for 3 colour channels.

$$Q_{z,\theta} = \sum_{c=1}^3 \left(\frac{T_{z,\theta,c} - \mu_{z,\theta,c}}{\sigma_{z,\theta,c}} \right)^2 \quad (3.60)$$

The texture image is an array of TPC denoted by $T_{z,\theta}$, where the coordinate of the texture image is (z, θ) . $\mu_{z,\theta,c}$ and $\sigma_{z,\theta,c}$ are a mean and a variance of TPC at (z, θ) of a colour channel c . The means and the variances are computed by the MOG method[83].

See the Chi-square test in Section 3.1.2. From the Chi-square distribution, the threshold $Q = 2.37$ that makes 50% of observations having lesser Q than the threshold and the other 50% having larger. Setting the threshold at $Q = 2.37$ makes the number of matching pixels and mismatching equal. A distracted particle state will have the number of mismatch TPC, which decreases the likelihood. The particle filter will remove the distracted hypothesis that generates the low likelihood and pushes the tracker back to the correct position.

Before we are moving to the conclusion of the algorithm, it is necessary to explain the data structure of the state. Figure 3.13 shows the data structure of the multiple targets state, where a particle state is a concatenation of state vectors in multiple rows. The column expresses the global state of all subjects being tracked in the scene.

$S =$

k	State	n			
		1	2	...	n_{max}
1	posx				
	posy				
	velx				
	vely				
2	posx				
	posy				
	velx				
	vely				
⋮					

\uparrow
 $S_{(n=1, k=2)}$

FIGURE 3.13: Data structure of the multiple target state S .

From Equation (3.58), we can re-express the silhouette log-likelihood in position region as Equation (3.61) and Equation (3.62). The texture log-likelihood is denoted by

Γ^3 in Equation (3.63).

$$\Gamma^a = \frac{1}{\pi ab} \sum_i \frac{1}{m_i} \left\{ I^f(p) > 1 \right\} \{0 < \psi(p_i, S)\} \quad (3.61)$$

$$\Gamma^b = \frac{1}{\pi ab} \sum_i \frac{1}{m_i} \left\{ I^f(p) > 1 \right\} \{-0.5 < \psi(p_i, S) \leq 0\} \quad (3.62)$$

$$\Gamma^c = \frac{1}{\pi ab} \sum_i \{Q_{z,\theta} \leq 2.37\} - \{2.37 < Q_{z,\theta}\} \quad (3.63)$$

The silhouette positive fill-in ratio (Γ^a), the silhouette positive fill-in ratio (Γ^b) and texture matching ratio (Γ^c) are normalised by the area of the ellipse. In order to combine silhouette and texture scores, we used a *logistic function*, [97] page 725. The new likelihood differs from the conventional method that was described in Section 2.5.4, which usually uses an exponential function to combine several measurements to a single value. There are two good reasons to use the logistic function instead of exponential function. First, the logistic gives output in the range -1 to 1, which does not explode as the exponential function. The conventional method computes the weight from exponential of similarity (or minus of dissimilarity). When the similarity score is large, which could happen when $\Gamma^a > \Gamma^b$, the likelihood weight will be larger than 1 and lead to unsuitability of the re-sampling method. We have to control the likelihood weight in the range between 0 to 1. In this case, the logistic offers the better stability. The second reason is that the exponential function is highly sensitive to tiny change in coefficients of the linear combination. That leads to difficulty of adjusting or learning the parameters. The logistic function also has been applied for combining signals in the artificial neuron network modeling because of smoothness and stability [165]. Equation (3.64) shows a calculation of the final likelihood (or the particle weight).

$$w = \text{logistic}(c_0 + c_1\Gamma^a + c_2\Gamma^b + c_3\Gamma^c) \quad (3.64)$$

$$\text{logistic}(x) = \frac{1}{1 + e^{-x}} \quad (3.65)$$

Algorithm 3.4 concludes our likelihood computation. It starts with loading camera parameters to local memory. Next the nested for loops of n, p and k are proceeded. The conic \mathbf{E} is generated for each subject and each particle. The ellipse parameters are computed from the conic and stored in local memory. Then ray tracing computation for every active foreground pixel is calculated. The number of pixels in negative region

ALGORITHM 3.4: Ellipsoid Likelihood

```

1  Input:    $I$  ;a colour image,
2             $I^f$  ;the foreground image and
3             $S = S_{n,k}$  ;a particles set
4  Output:  $w = w_{n,k}$  ;a set of output weight
5  Load camera parameters  $(\mathbf{KR})^{-1}$  and  $e$ 
6  For each particle  $n$ 
7       $\mathbf{E} \leftarrow \text{conic}((\mathbf{KR})^{-1}, e, S_n)$  ;computing conic from Eq. 3.33
8       $(A, B, C, c_u, c_v, a, b) \leftarrow \text{ellipse}(\mathbf{E})$  ;computing ellipse parameters
9       $\Gamma_k^a \leftarrow 0$  ;a counter of positive region
10      $\Gamma_k^b \leftarrow 0$  ;a counter of negative region
11      $\Gamma_k^c \leftarrow 0$  ;a counter of texture matching
12     For each pixel  $p$ 
13         If  $I^f(p) = 1$ 
14              $m \leftarrow 0$  ;number of overlaid ellipses on the pixel  $p$ 
15             For each subject  $k$ 
16                 compute rectangular bounding box from radii  $a$  and  $b$ 
17                 If the pixel is in the bounding box
18                      $h \leftarrow \psi(p)$  ;computing contour level from Equation (3.37)
19                     If  $h < 0$ 
20                          $\Gamma_k^b \leftarrow \Gamma_k^b + 1$ 
21                     Else
22                          $m \leftarrow m + 1$ 
23                         compute texture coordinate  $(z, \theta)$ , from Section 3.2.2
24                         compute  $Q$  by Eq.3.60
25                         If  $Q < 2.37$ 
26                              $\Gamma_k^c \leftarrow \Gamma_k^c + 1$ 
27                         Else
28                              $\Gamma_k^c \leftarrow \Gamma_k^c - 1$ 
29                         End
30                     End
31             End
32         End
33         For each subject  $k$ 
34             If  $0 < h$ 
35                  $\Gamma_k^a \leftarrow \Gamma_k^a + \frac{1}{m}$ 
36             End
37         End
38     End
39 End
40 For each subject  $k$ 
41     area  $\leftarrow \pi ab$ 
42      $\Gamma_k^a \leftarrow \Gamma_k^a / \text{area}$ 
43      $\Gamma_k^b \leftarrow \Gamma_k^b / \text{area}$ 
44      $\Gamma_k^c \leftarrow \Gamma_k^c / \text{area}$ 
45      $w_{n,k} \leftarrow \text{logistic}(c_0 + c_1 \Gamma_k^a + c_2 \Gamma_k^b + c_3 \Gamma_k^c)$  ;Equation (3.64)
46 End
47 End

```

Γ_k^b of the subject k is counted. The number of pixels that matched the texture image is counted and saved into Γ_k^c . And the number of the overlaid ellipses Γ_k^a is also calculated. Finally, all counters are combined into a single value by the logistic function and stored as the particle weight \mathbf{w} .

3.4 Our Particle Filter for Multiple Target Tracking

In this section, we will emphasis on our particle filter framework which has been proposed to solve two major dilemmas in multiple targets tracking application: distraction problem, which can be solved by an improve likelihood function and the short-time disappearing problem that can cause failure in tracking. Our objective is to bring about a practical system that can process in real-time and give adequate accuracy (MOTA around 80%).

A particle state is a global state of all subjects expressed by a concatenation column vector. The process in the likelihood function computes all subjects, which are represented by a particle, at the same time. The ownership of a pixels computed from overlaid ellipses that makes the multiple subject state no longer completely independent.

3.4.1 Likelihood with distraction suppression

In [19], Kreucher compared between applying joint state and independent state in the multiple targets tracking problem.

The independent state represents each subject independently and the likelihood is also computed independently. The independent Bayesian is accurate when subjects are separate in the observation space, however, it suffer from severe distraction when subjects are overlaid in the observation space.

In contrast, the joint state expresses all subjects by a single long vector and computes the likelihood globally. When the state vector is joined, the number of dimension of state space increases and leads to an explosion of the searching volume in the state space. The problem is also known as *the curse of dimensionality*[166]. So, the joint state particle filter requires more particles to cover the enlarged searching volume to

maintain the same accuracy. It is impractical to process in real-time. Kreucher suggested an alternative method where the joint likelihood computation is necessary only when targets are interacting with each other. The interaction can be easily detected by determining the Euclidean distance between them. In our method we detect interaction in the observation space (image coordinate) by checking the intersection of the ellipses contours and using ownership computation.

The independent particle filter faces a major problem of distraction but for the joint state this is unsolvable in real-time. If a tracking system is unable to perform in real-time it could become useless in many applications. So we gave first priority to the speed. One solution to solve the distraction problem is by including more useful features in the state vector. Increasing the independent state helps the tracker to identify the subjects. With additional features, an extra searching space is added and they slow down the process but accuracy is improved significantly. In our tracking, the texture image of the subject is acquired during tracking which is a unique feature that help to solve the distraction problem. The texture image is initialised at the beginning period of tracking, during this period the distraction suppression is a key to reducing the distraction rate at early period of tracking. After the texture image is established by the learning process the texture likelihood can contribute to the likelihood function.

Our strategy is to use the fast independent particle filter and attempting to prevent distraction by improving effectiveness of the likelihood function. We expected high precision when subjects are separate as a major advantage of the independent state over the joint state. When the subjects are in close proximity the distraction suppression must prevent the problem for a sufficient time to allow the tracking system to obtain the texture image. Once the texture image is ready, the distraction rate is very low because the texture of each person is distinctive.

3.4.2 Visibility state

The next problem is the short-time disappearing caused by occlusion. In a practical scenario, some static objects such as a small tree or moving car can cause short-time disappearing. The occlusion prevents our observation while the subject is still moving behind the obstacle. Lack of observation increases the uncertainty of estimation over time because an unknown state keeps changing. The occlusion can also cause distraction

when a subject A is moving just behind an obstacle and another subject B is passing nearby; the tracker of the subject A can be distracted and follow the subject B. Our strategy is to let the subject A become invisible and give a small likelihood to the tracker to prevent the distraction. So an extra variable, which is called *visibility*, has been added into the state vector.

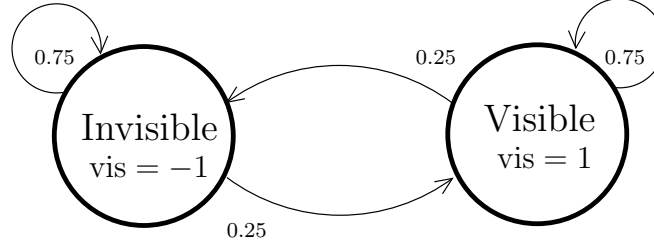


FIGURE 3.14: Markov Chain Model of visibility state of a subject

Figure 3.14 shows a Markov Chain model of the visibility state of a subject, which can either stay in same state with probability 0.75 or turn to an opposite state by 0.25 of probability. A state variable **vis** is in set $\{-1, 1\}$ the visibility is used in order to compute *persistence level* **per**. The transition of the visibility Markov Chain model and persistence are expressed in equations below.

$$\mathbf{vis}_t = \mathbf{vis}_{t-1} \times b \quad ; P(b = 1) = 0.75; P(b = -1) = 0.25 \quad (3.66)$$

$$\mathbf{per}_t = \mathbf{per}_{t-1} + \mathbf{vis}_{t-1} \quad (3.67)$$

Therefore, we can estimate disappearing time and allow the tracker to keep the previous motion. Once, the persistence level drops below the threshold, the system will decide to terminate the tracker.

3.5 Summary

From our primary tracking based on the quadrilateral silhouette model, we modified the likelihood from a simple rectangular to a parametric ellipsoid projection method. The ellipsoid projection method enables detection and tracking to be performed in 3D instead of the conventional 2D method. The parametric ellipse function improves the computation speed by reducing pixel-level calculations and minimizing memory utilisation to store the human model. The parametric ellipsoid approach significantly changes

the way to compute the likelihood, where the parametric boundary can be computed relatively faster than the vertex base model as estimated in Table 3.2. In addition, the ellipsoid projection allows the system to obtain a texture image around a human body and makes texture similarity measurement possible. The texture image expresses a unique state and avoids the distraction problem.

To obtain the texture image the tracker must follow a correct target with high accuracy at an early stage of tracking. High accuracy tracking during the early stage is achieved by the ownership model, which is computed from ellipses overlapping. This method is also call distraction suppression.

In Chapter 4, a sequential implementation and evaluation of the detection-tracking system is discussed in detail. The parallel algorithm of the system is explained and compared in Chapter 5.

Chapter 4

Sequential Implementation

In this chapter, we discuss the design and implementation of the tracking algorithm on a standard CPU. Normally, a tracking program starts by manually selecting a target as in our single tracking system as showed in Section 3.1 that makes the system not fully automatic. So the system should consist of both detection and tracking modules. Both modules are developed based on the ellipsoid model as described in Chapter 3 that allows many cameras to work together as a unit. In this Chapter, at the beginning we consider these two modules individually and later they are combined into single system. In order to make the tracking system perform fully automatically without human intervention, the system must initiate tracking by exploiting object detection. Our detection method will be described in Section 4.1.1. Our multiple people tracking system has been built carefully by combining several methods, the tracking, the detection method and extra strategies, as discussed in Chapter 3. The design is transferred to a computer program and evaluated using various datasets including our own dataset. There are calibration parameters available for all benchmark datasets that we are using. So we can determine the subject position in 3D and measure the accuracy of the tracking framework. This chapter consists of algorithm design, sequential implementation for a standard CPU, performance evaluation and speed profiling . In Chapter 5 we will compare the controlled sequential implementation with a parallel implementation on a GPU (Graphics Processing Unit).

4.1 Multiple Target Detection and Tracking Framework

The framework consists of detection and tracking modules that we will divide them and consider separately. In the detection module, the original image from each camera is processed to produce a foreground image and find a new subject. Simultaneously, the tracking module continually estimates the state of the subject immediately after a new subject is added until the subject is removed by persistence evaluation. A tracker is activated by the newly detected subject. The entire process starts with the detection function.

4.1.1 Detection

The detection function is based on a grid response, where many grids are located on the ground floor: the collection of locations or *grids* is denoted by G . An ellipsoid model with a vertical diameter of 1.7m and a horizontal diameter of 0.5m is placed on each grid. Then the ellipse parameters are computed and a rectangular bounding box is calculated for each ellipse. For each ellipse we need 4 corners to express the rectangular bounding box. The set of all bounding boxes for all ellipses is denoted by R . The preparation of R is done during initialisation at the beginning of the overall process.

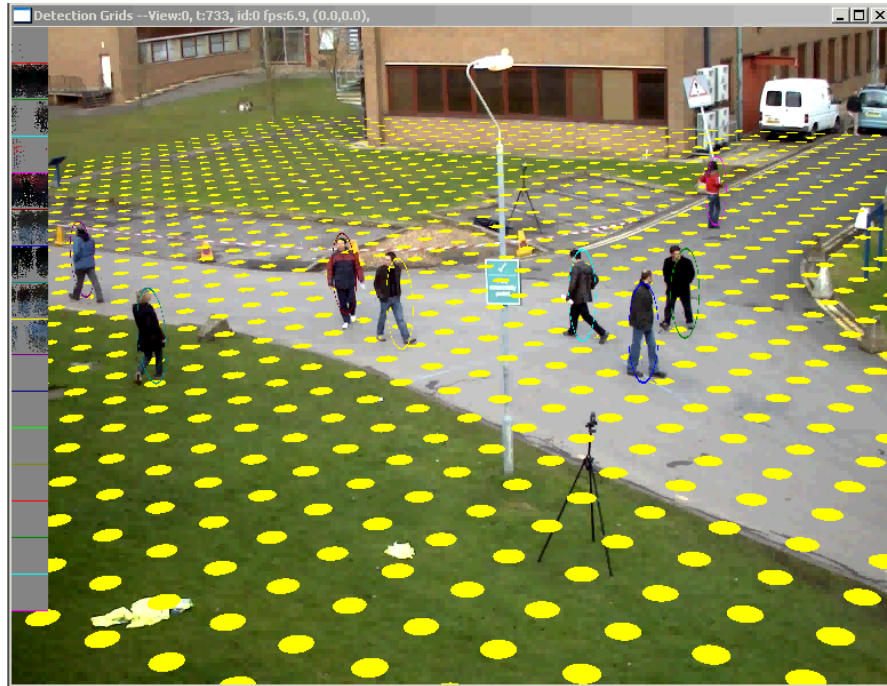


FIGURE 4.1: Detection grids (yellow circles) and ellipsoids projection.

Figure 4.2 shows the people detection module. For every frame, the original image is captured and the MoG background model is estimated. Then foreground pixels are extracted. Next the module computes the integral image J as described in [59, 167]. Using the integral image for kernel integration accelerates the method by pre-computing all summations and then we can calculate the kernel integral at a specific location by using the integral image as the lookup table. The pixel coordinates of top-left, top-right, bottom-left and bottom-right of the rectangular boundary are denoted by $r = \{p_1, p_2, p_3, p_4\}$ and $r \in R$, respectively.

$$w_{g \in G} = \frac{J(p_1) - J(p_2) - J(p_3) + J(p_4)}{A_{rec}} \quad (4.1)$$

Where A_{rec} is the area of the rectangular detection bounding box and the integral image is computed from the equation below.

$$J(p_x, p_y) = \sum_{y=1}^{p_y} \sum_{x=1}^{p_x} I^f(p_x, p_y) \quad (4.2)$$

The response weights $\{w_g; g = 1, 2, \dots\}$ of all grids for every camera are computed. Any low response weights, which are less than a threshold of 0.65, are removed and a grid with a global maximum response weight will be selected as a new target. The selected position will activate a tracker. Only one subject can be added in a single time frame to reduce the false alarm rate.

In order to activate a tracker all particles must be initialised. The initialisation defines the prior density of the state of the new subject for the first time. Then the transition function in the particle filter will generate the prior in the subsequent iteration. The prior density of position is assumed to be a uniform distribution within a 1m radius on the ground plane. The prior density of velocity is also drawn from a uniform distribution with magnitude between 0 and 2ms^{-1} in all directions. Visibility is set to 1 and the persistence level is maximum. The first generation of particles is evaluated by the likelihood function and then the subject will be handled by the tracking module.

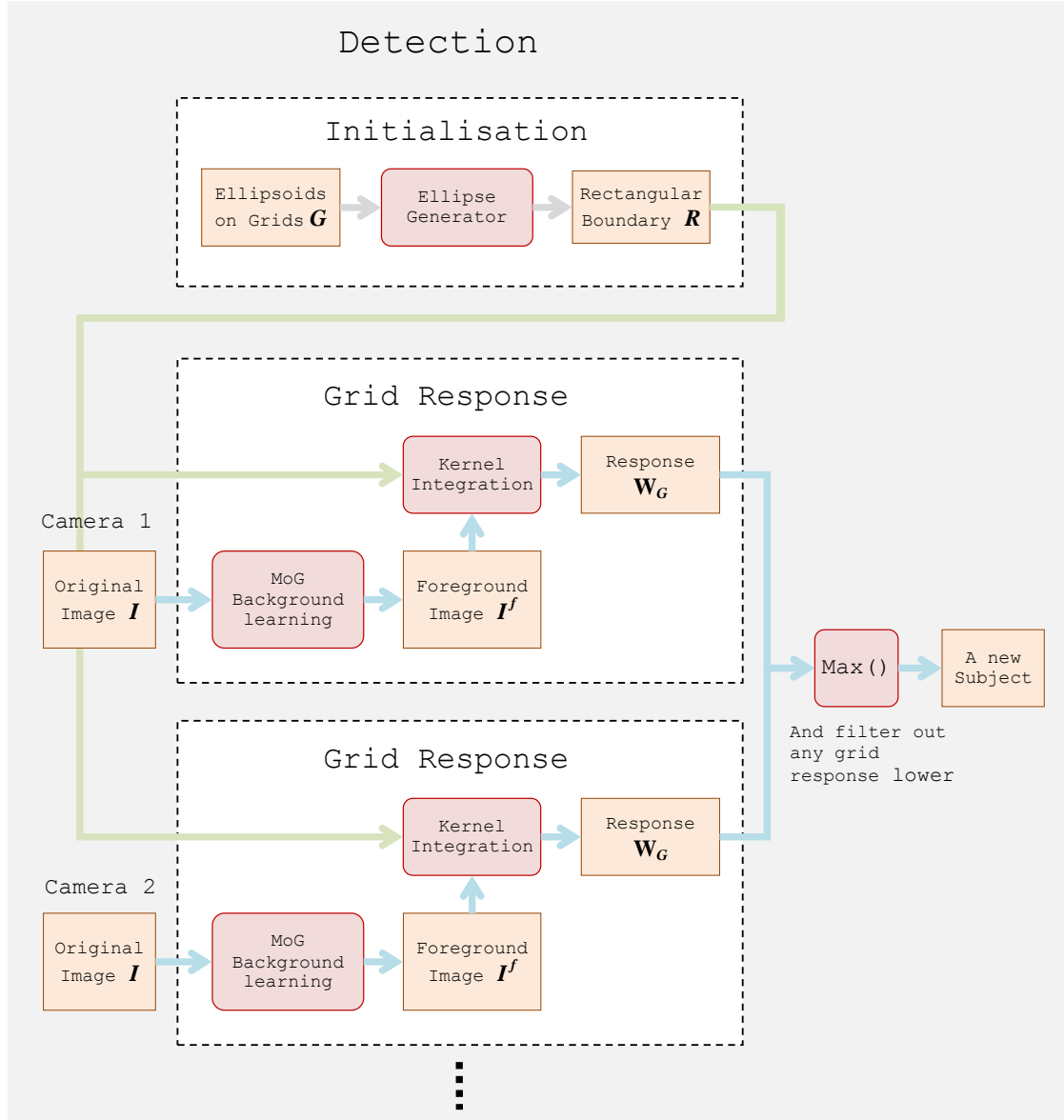


FIGURE 4.2: Detection function. The rectangles and the round shapes are data and operations, respectively

4.1.2 Tracking

Figure 4.3 is our modification of the SIR particle filter. The original SIR particle filter has three components (likelihood, re-sampling and transition). In our framework the likelihood involves calculation from many cameras. The likelihood weights from all cameras are combined by the belief function. A subject that is visible in a camera may not be visible to other cameras due to occlusion and limited field of view. Observations from different cameras can cause conflict when an occlusion happens. So, combining all likelihoods in the fusion function must consider the reliability from each camera. We

subject from visible to invisible.

$$w'_{e,k,n} = \begin{cases} w_{e,n,k} & ; \text{if } (\bar{w}_{e,k} > \epsilon) \\ 1 & ; \text{otherwise} \end{cases} \quad (4.4)$$

$$\bar{w}_{e,k} = \frac{1}{N} \sum_{n=1}^N w_{e,n,k} \quad (4.5)$$

In Figure 4.3, the filtered likelihood is combined using Equation (4.3). The combined likelihood weights are passed to re-sampling and transition functions as same as in a standard SIR particle filter. The details of the re-sampling and transition functions have been described in Chapter 3. `Transition()` generates a new set of particles that represent the prior density. In `Remove()`, the existence of a subject is decided by considering the expectation persistence state. To avoid multiple-trackers focusing on the same subjects, if any trackers occupy same space, the subject which has lower persistence will be penalised by reducing the persistence. Therefore, the persistence can be reduced by both invisibility mechanism and duplication of trackers. If the persistence is lower than 0, the subject will be terminated and the corresponding tracker will be deactivated (sleep mode). The next step of the process is `Add()` function, which listens for a message from `Detection()`. If there is a new subject to be added, the `Add()` function will initialise the particles. Finally, the texture model of each subject is updated using the previous texture image and the current ray tracing observation as described in Section 3.3.3. Then the process repeats by computing the likelihood again.

4.2 Sequential Algorithm

In conventional algorithm design, the algorithm is a sequence of instructions. A processing unit processes each instruction one-by-one until the end of the program. The reason for processing in sequence was that there was only a single processor that could perform a single instruction at a particular time. A single worker can do a single task in a time frame. However, the particle filter expresses the probability function by many of particles. So, the processor has to perform similar tasks repeatedly in a sequential mode.

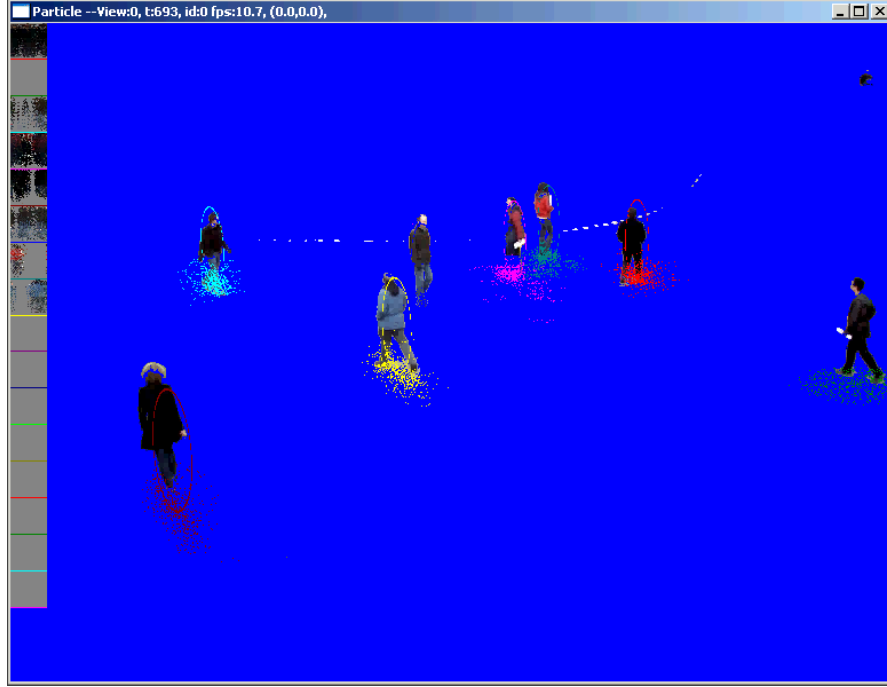


FIGURE 4.4: Background segmentation and particles on ground floor.

From summarized diagrams in Figure 4.2 and 4.3, the red round blocks are operations and the orange sharp blocks are stored data in memory. We can transform the diagrams to Figure 4.5. The algorithm starts by initialising the grids, ellipses and tracking parameters, then reading an image from a camera. The process computes the detection function, which consists of MoG background learning and an kernel image integration sub-function as shown in Figure 4.2. The colour image and the foreground image are used as the input to the likelihood function. `Read()`, `Detection()` and `Likelihood()` are repeated for every camera in the network. Next the likelihood weights are combined in the `Fusion()` function, followed by the standard functions of the SIR particle filter `Re-sampling()` and `Transition()`. `Remove()`, `Add()` and `UpdateTexture()` are called before finishing the iteration of the current frame and then calculations are repeated after reading images in the new frame. Each function block is constructed from internal for-loops, which are processed sequentially.

4.2.1 Implementation

We implemented the sequential algorithm in C++ and used OpenCV in order to read and display the output. The algorithms are fairly complex so we split the system into several classes for the benefits of programming.

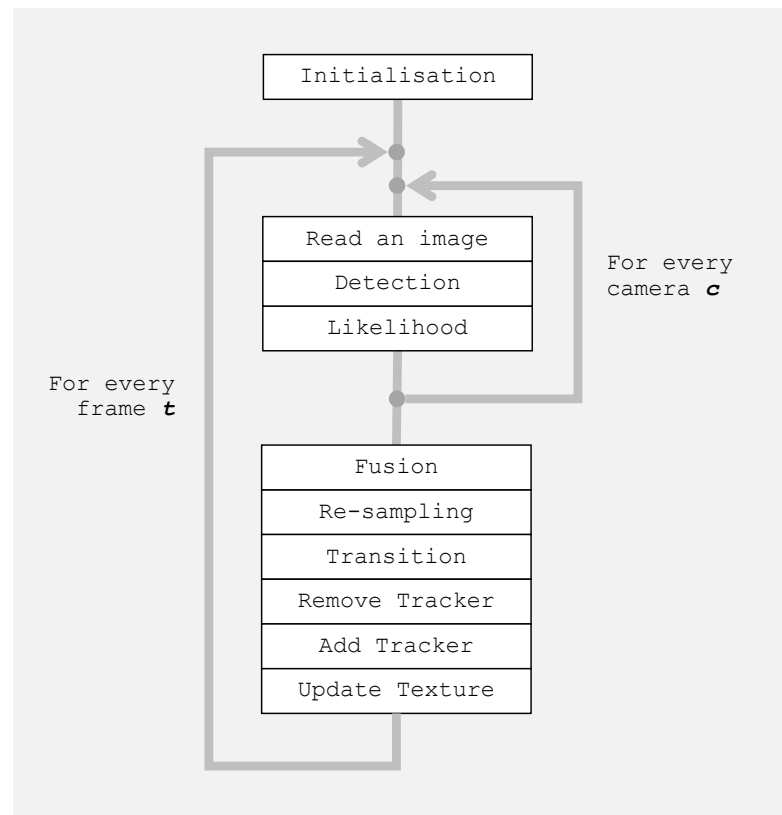


FIGURE 4.5: A sequential algorithm of the detection tracking system.

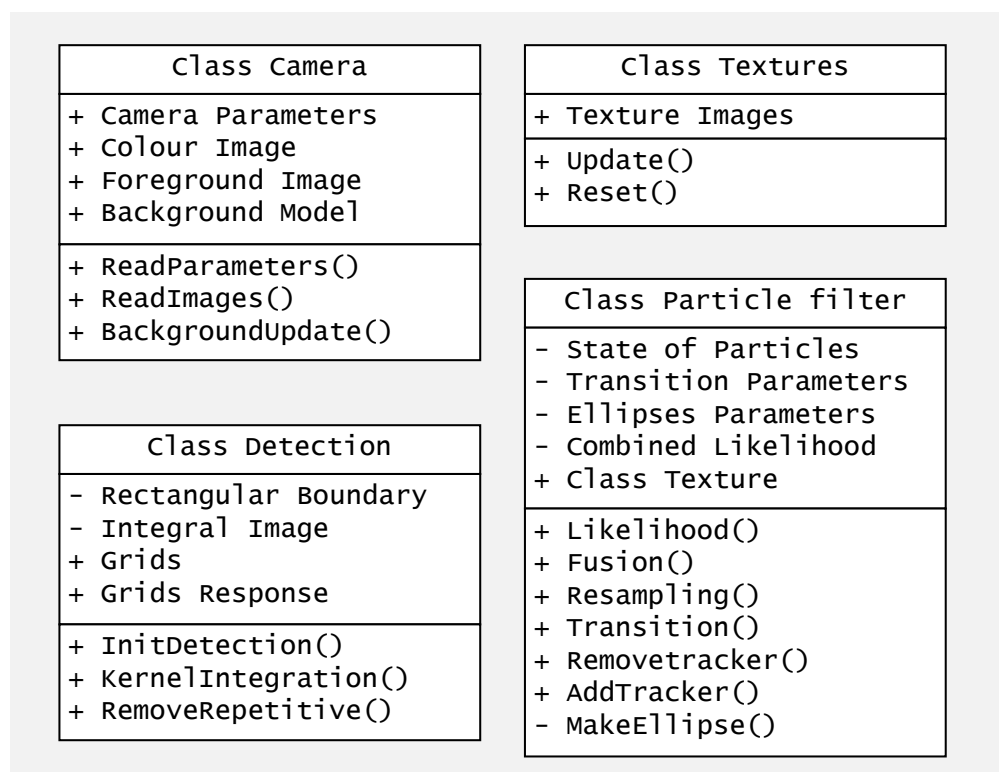


FIGURE 4.6: A class diagram of the detection tracking program.

Figure 4.6 shows 4 major classes of the program. The Camera class contains data members such as camera geometric parameters, images and background model. Function members in the camera class are `ReadParameters()` and `ReadImage()` from the local hard drive or a camera. The Camera class also has `BackgroundUpdate()` function. The Detection class focuses on calculating the grid detection response weights from kernel integration. The detection function could detect a currently tracked subject, so `RemoveRepetitive()` is a function to remove repeated detection (duplication trackers) of a current tracked subject. The particle filter class is the most complex class including `Likelihood()`, `Fusion()`, `Resampling()`, `Transition()`, `Remove()` and `Add()`. The computation involves the state of the subjects. `MakeEllipse()` is a private function to calculate ellipse parameters from an ellipsoid model. The texture class consists of a data member of texture images and two public function members, `Reset()` and `Update()`.

4.3 Experiment on Sequential Implementation

In this section, we discuss the precision and computation time of the system. The detection and tracking frameworks which are described in this chapter can be implemented as sequential or parallel versions. A correct implementation must have identical results in both versions. The speed profile results in this section will be compared with a parallel implementation in Chapter 5. Both sequential and parallel systems have been tested with the PETS09 dataset [160]. The Multiple Object Tracking Accuracy (MOTA) in [168] is applied for evaluating our technique.

The MOTA is computed from the summation for all frames t of f_t false alarm, m_t miss detection, s_t switch events and divided by g_t total number of subjects in the reference view. The MOTA is defined by the equation below.

$$MOTA = \left(1 - \frac{\sum_t f_t + m_t + s_t}{\sum_t g_t} \right) \times 100\% \quad (4.6)$$

Figure 4.7 shows three types of failures in multi-target tracking that commonly happen and are counted in the MOTA score. When a tracker is away from its subject by more than 1m the miss detection and false alarm are counted.

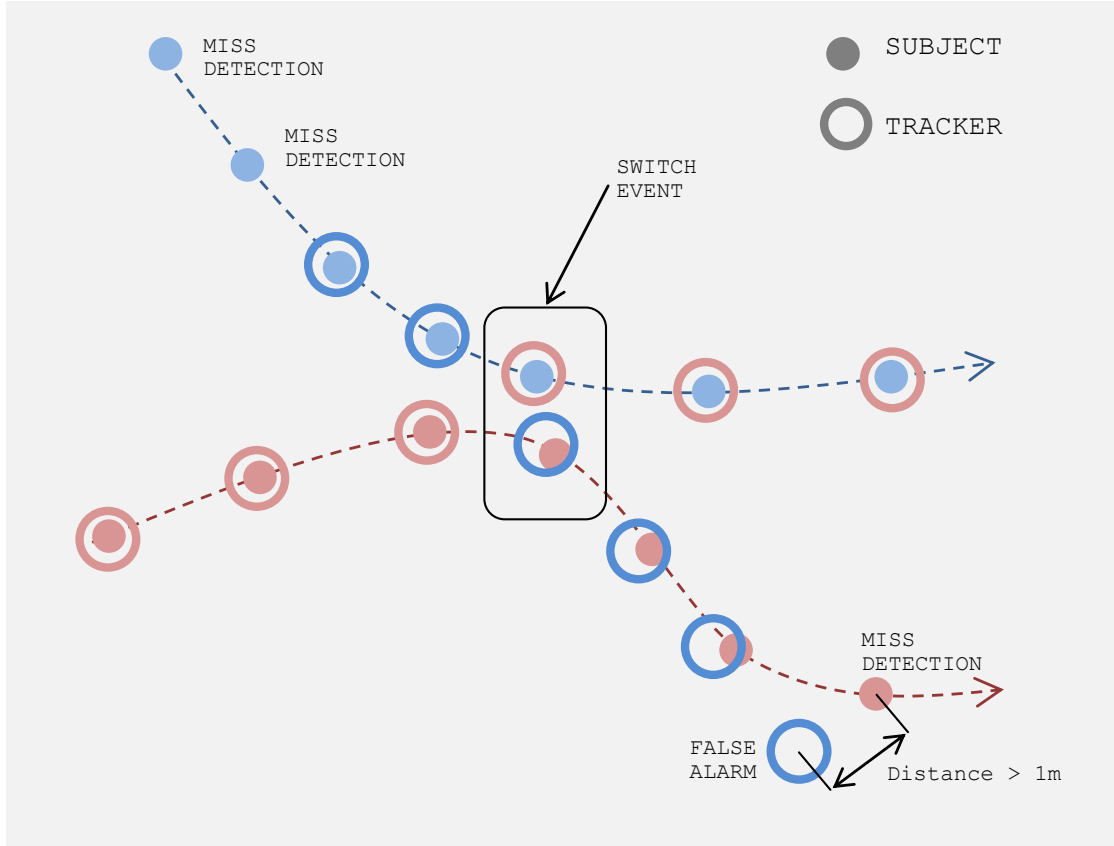


FIGURE 4.7: Events in MOTA , miss detection, false alarm and switch events.

4.3.1 Advantage of using texture similarity

In this section we compare the accuracy of two systems, where one uses texture in likelihood computation and another does not. This configuration can be done by setting the texture likelihood coefficient in the Equation (3.64). When the texture was used, the coefficients in Equation (3.64) were $c_0 = -8$, $c_1 = 20$, $c_2 = -5$ and $c_3 = 50$. And c_3 was set to 0 when texture information was excluded. The coefficients were selected by finding the optimal values that yield the highest MOTA score evaluated using the PETS09 dataset and we kept using these values in all datasets.

The image sequence from the first and third cameras were used in this texture test. All camera parameters from PETS09 in the xml files were exploited to get correct camera parameters. The rotation matrices were computed by the Euler angles in the xml files and Equation (3.10). The invisibility score in our tracking was set to 0.15 and the likelihood fusion threshold was 0.01.

Because the sequence of images for PETS09 is short compared to the background learning period, which required about a hundred images, we extended the sequence by running it backward from frame 794 to frame 0 to initiate the background model, then started the tracking and recording the estimated state variables from frame 0. The output state estimation is saved in a xml for every frame for further evaluation. There were 795 frames and each frame has a collection of estimations of all current existing subjects. The estimation was compared with ground truth, which was created manually by inverse projection from the image coordinate to the ground plane coordinate system. The evaluation method is similar to Figure 3.6.

The detection and tracking program was tested in two modes (with texture and without texture) to see the effect of the texture similarity on tracking performance. Each mode of tracking was repeated 10 times on the same input image sequence. Each time the random seed number was set to the machine clock number. The Monte Carlo method is sensitive to a random seed number so each test must have a different result. Finally, The MOTA was computed statistically.

We have a small problem of uneven ground plane in PETS09 in the grass area. The grass area is higher than the ground plane that was used in calibration. In Figure 4.8, the projected ellipses of the lady on the green grass area in camera1 and camera3 are inconsistent as the grass surface is not on the ground plane. The white balance of images from camera3 were adjusted to make the colour consistent in both cameras.

TABLE 4.1: Comparing using and not using the texture similarity

	$f(\%)$	$m(\%)$	$s(\%)$	MOTA(%)
with texture	5.20 ± 3.64	3.76 ± 1.48	0.26 ± 0.14	90.7
no texture	7.16 ± 5.33	5.83 ± 2.60	0.46 ± 0.37	86.5

Table 4.1 shows the advantage of using texture. We assumed the distribution of the error in Table 4.1 is normally distributed. The uncertainty or the standard deviation is denoted by the number after the symbol \pm . The mean and the standard deviation are computed from 10 times repeated experiments. In this set of experiments we used 0.5m for the detection grid spacing and $(a_x, a_y, a_z) = (0.5\text{m}, 0.5\text{m}, 1.7\text{m})$ for the ellipsoid diameters. The overall MOTA result from tracking with texture is 4% better (but not statistically significant at p-value 0.05). The texture similarity prevents the tracker drifting away from the subject leading to a decrease in false alarm rate and miss detection



FIGURE 4.8: Evaluating with PET09 dataset. First and second columns are result sequences from camera1 and camera3, from top to bottom are frame 700th, 720th and 740th. The numbers over ellipse show ID and height in unit meter.

rate. In this experiment, the miss detection rate was significantly improved by the texture likelihood (at p-value 0.05).

4.3.2 Accuracy evaluation

Reduction of the number of particles affects the precision and accuracy of the tracking performance. We examined the effect by varying the number of particles from 128 to 256, 512 and 1024. We tested the tracking framework on 4 different datasets, PETS09,

PETS06, PETS03 and EM330 (the last dataset was captured in our lab). In this set of experiments we used 1.0m for the detection grid spacing and $(a_x, a_y, a_z)=(0.5\text{m}, 0.5\text{m}, 1.7\text{m})$ for the ellipsoid diameters.

TABLE 4.2: Error and number of particles.

Dataset	n_{max}	$f(\%)$	$m(\%)$	$s(\%)$	MOTA	j_{est}
PETS09	128	1.178	4.367	0.02	94.435	45
$\sum g_t = 4923$	256	1.300	3.555	0.05	95.095	45
$j_{true} = 17$	512	0.934	5.139	0.02	93.907	37
	1024	0.914	4.753	0.04	94.293	37
PETS06	128	3.853	3.395	0.06	92.692	21
$\sum g_t = 4801$	256	3.749	3.249	0.06	92.942	20
$j_{true} = 17$	512	2.958	3.791	0.06	93.191	19
	1024	3.395	3.812	0.06	92.733	19
PET03	128	0.210	5.672	0.03	94.088	71
$\sum g_t = 36194$	256	0.262	5.313	0.02	94.405	66
$j_{true} = 33$	512	0.202	5.280	0.01	94.508	64
	1024	0.262	5.274	0.02	94.444	62
EM330	128	9.024	15.311	0.15	75.515	19
$\sum g_t = 1352$	256	9.024	12.796	0.15	78.030	17
$j_{true} = 6$	512	9.024	8.580	0.07	82.326	16
	1024	9.024	15.163	0.00	75.813	17

Table 4.2 shows the results from the experiments. When the number of particles was increased, the false alarm rate (f) was decreased in 3 datasets but in our dataset EM330 the false alarm rate is constant. The miss detection rate (m) was almost constant relative to the growth in number of particles. In our opinion the miss detection rate reflects the effectiveness of the detection module with only slightly effect from the tracking module because the detection works independently from the particle filter. So varying the number of particles should not affect the detection performance.

The switch event rate (s) is slightly low compared to the false alarm and missed detection rates. Our framework tends to terminate a tracker before the switch event happens. The removing mechanism breaks a trajectory into many sub-trajectories and the switch event cannot report the errors. The number of sub-trajectories (j_{est}) estimated by the framework has been measured and shown in the last column, whilst the true number of trajectories (j_{true}) is noted in the first column. The number of estimated sub-trajectories (j_{est}) should be close to the true number (j_{true}). The correctness

of the number of trajectories implies accuracy of the tracking system to follow the same subject and it should be included in the multiple object tracking accuracy metric. The correctness of the j_{est} was improved slightly by increasing the number of particles.

There were small improvements caused by increasing the number of particles. However, the crucial problem was that the system could not compute fast enough to return an output before the next input. An increase in the number of particles can slightly improve the accuracy but it may prevent the system running in real-time. In the next section, a study of computing time that is affected by increasing the number of particles will be discussed.

Figure 4.9 shows screen captures of the tracking results using 1024 particles. At the middle of the scene the static sign post blocks the camera visibility. Subjects passing behind the sign post could cause tracking failure. For example the green and brown trackers can manage to track under the occlusion. However, our tracking framework includes invisibility modeling that allows short-period disappearance and it can track those targets with a low chance of failure.

Figure 4.10 shows mutual occlusion between two subjects. This situation normally causes severe distraction. A dominant subject draws attention from another tracker and leads to failure. Our tracking system integrates texture similarity, so each tracker knows the unique corresponding subject.

Our system can scale the number of targets up to several subjects. Figure 4.11 shows results from tracking a football match in PETS03. Tracking football players is relatively easier than tracking people on a street because the clear green grass in background makes background segmentation almost perfect.

Because the ellipses are derived from the 3D ellipsoids, our tracking system can perform tracking for very close or very far subjects reliably. Figure 4.12 shows the perspective scaling effect and our tracking system performance is still good.



FIGURE 4.9: From top-left to bottom-right are frames 135th to 175th from PETS09.

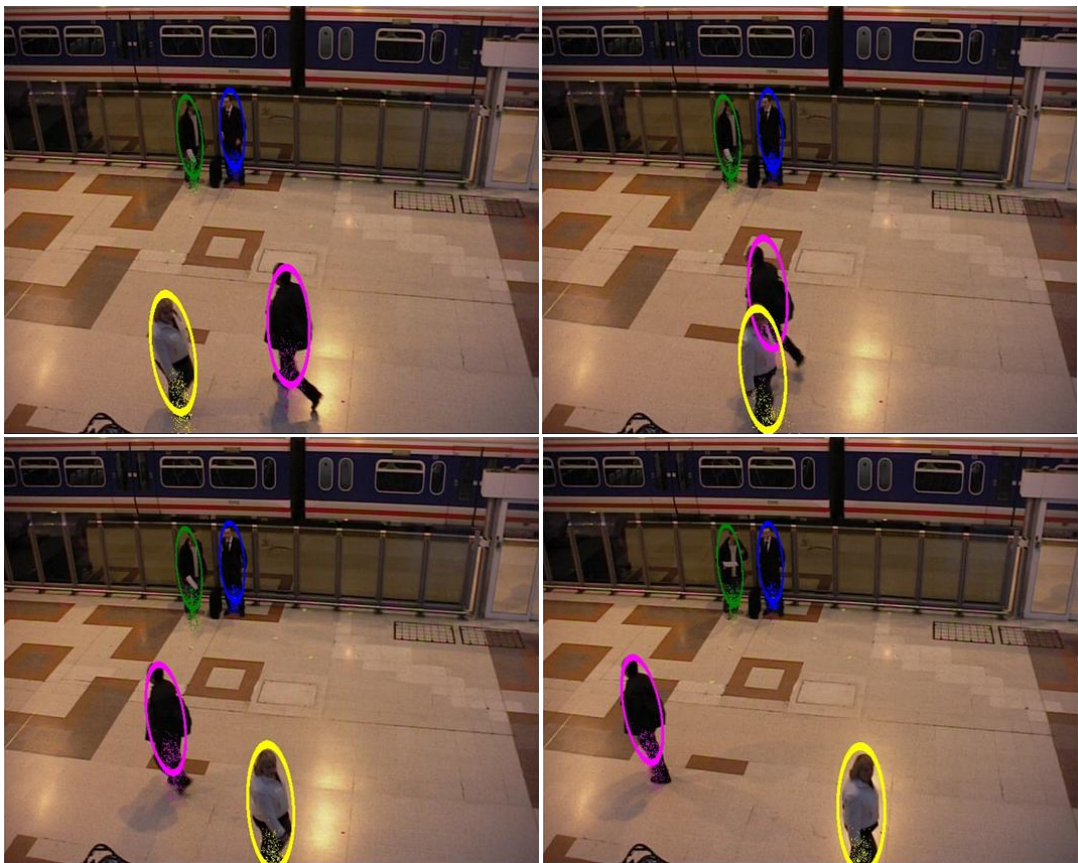


FIGURE 4.10: From top-left to bottom-right are frames 1500th to 1560th from PETS06.



FIGURE 4.11: From top-left to bottom-right are frames 1500th to 1560th from PETS03.

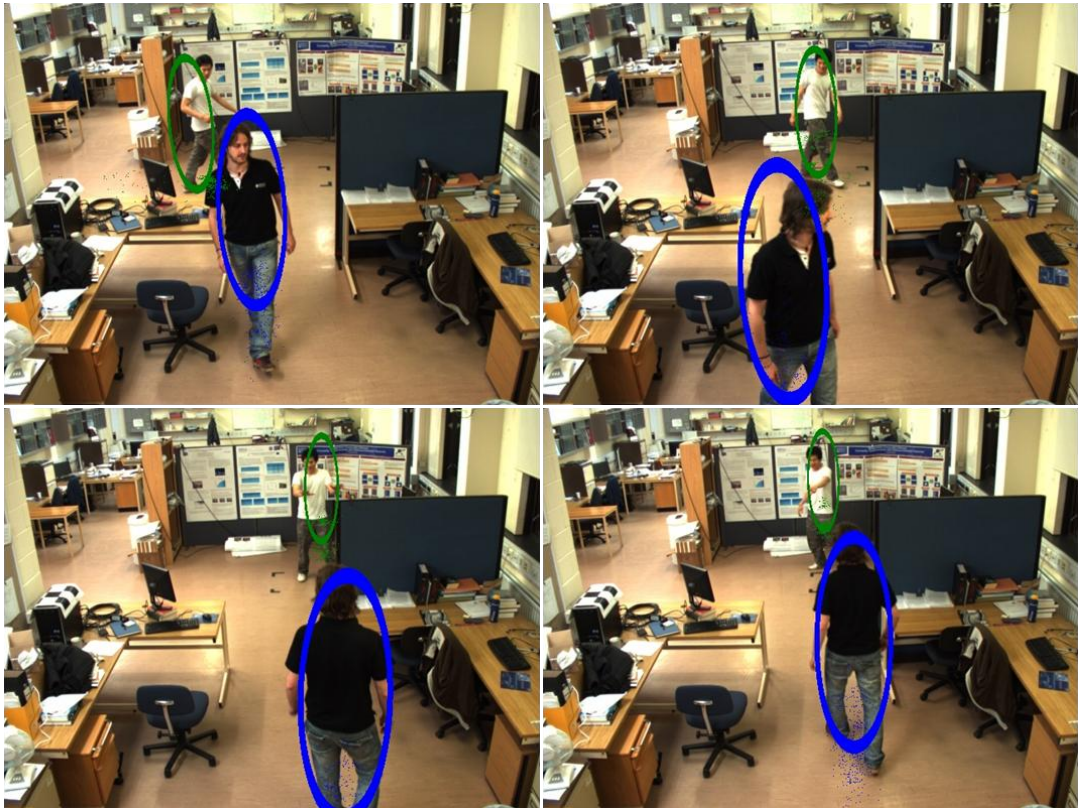


FIGURE 4.12: From top-left to bottom-right are frames 510th to 540th from EM330.

4.3.3 Computing time

The speed test or profiling of each function is the most important test for a real time application. In this section we consider the latency of each function and also study the effect of input on the total speed. We vary the number of particles to be used in tests. In each dataset, the number of targets and the number of active foreground pixels changing over time allows us to measure the relation between them and the computing time. This experiment has been tested on a work station computer with Intel Core i7 950 CPU at 3.07GHz. Measuring computing times with different machines always returns different speeds at it depends on both clock frequency and architecture.

```

1  #include <iostream>
2  #include <time.h>
3  int main() {
4      static ofstream table;
5      table.open("table.txt");
6      for (int t=0;t<tmax;t++) {
7          tick=GetTickCount();
8          function1();
9          table<<GetTickCount()-tick<<"\t";
10
11         tick=GetTickCount();
12         function2();
13         table<<GetTickCount()-tick<<"\n";
14     }
15 }

```

LISTING 4.1: An example code for profiling the `function1()` and `function2()`

The detection module consists of pixel-level functions such as background learning and integral image. As the detection module is constructed from repetitive simple instructions at pixel-level, so its complexity increases proportionally to the number of pixels and the number of grids being used.

In the tracking module the likelihood computation is the most complex sub-function. Over 80% of tracking computing time was spent on the likelihood function. The number of active pixels and number of targets are key factors in the computing time of likelihood because of the nested for-loops in the likelihood function. Table 4.3 shows the average time spent on each function of the tracking system. We used the timer function `GetTickCount()`, which has low resolution at 16ms per tick. Listing 4.1 shows how to measure the computation time of each function and save the output in a text file with

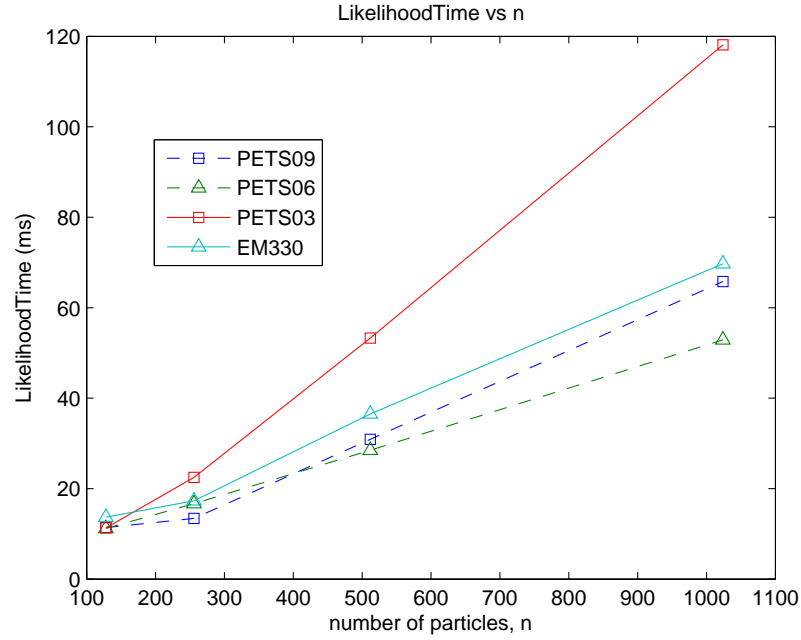


FIGURE 4.13: Relation between likelihood computing time and number of particles.

a tab delimiter table formate. Some functions such as `Fusion()` and `Remove()` used time smaller than the resolution, hence the computation time of those functions was measured as zero. Note that `Detection()` and `Likelihood()` are computed once per camera. If we have 4 cameras the functions will be called 4 times per iteration.

Table 4.3 shows the computation time per call of the framework. The computing time of detection is relatively constant with respect to the increase of the number of particles in all datasets. The likelihood computation time is approximately linear in proportion to the number of particles, as in Figure 4.13. The computing times of `Resampling()` and `Transition()` also rise when the number of particles increases.

Figure 4.14 shows the likelihood computation time of the experiments. The first column shows the linear relation between the likelihood computing time and the number of active foreground pixels (p_a). The graphs in the second column shows a linear relationship between the likelihood time and the number of subjects (k). From the nested for-loops structure in the likelihood function, we can conclude that the likelihood computing time increases when k, n or p_a increase and the nested for-loop structure in the algorithm suggests that the likelihood computing time can be expressed by the product

of k, n and p_a . Figure 4.14 shows a linear relationship between k, n and p_a .

$$\text{LikelihoodTime} \propto k \times n \times p_a \quad (4.7)$$

The result suggests that task parallelism can be done by dividing tasks according to either k, n or p_a . The parallel processing implementation is discussed in more details in Chapter 5.

4.4 Comparison

A comparison of MOTA performance is available in PETS09 workshop [153]. Originally, tracking algorithms considered only the position of detected subjects and overlooked the ability of the tracking system to retain the ID of the subjects to create complete trajectories. This ability is very important in multiple targets tracking problems. The ability to retain the ID of subjects was brought to attention in 2008 by introducing the MOTA[168]. Eventually, some researchers started to consider the ability to retain ID. The MOTA result from the PETS09 workshop is showed in Table 4.4. Note that Berclaz et al [101] gave the best MOTA (and other scores) from the evaluation [153].

Arsic [154] and Berclaz [101] used a detection method to transform images to a detected position in the 3D world coordinate system. Thus, the rich information of pattern and signature of subjects are overlooked. Breitenstein [157] included appearance classification in data association before performing final estimation by a particle filter. Including the appearance description exploits the advantage of subject signature. However, the 2D appearance descriptor is unable to be applied in multiple camera scenarios because different cameras generate different appearances. Thus, camera calibration is very important to make an accurate multiple camera tracking system. As in Table 4.4, using a calibrated camera method [156] can yield higher MOTA than non-calibrated method [157].

TABLE 4.3: Profiling of sequential implementation to measure computational time in millisecond evaluated from various datasets

Dataset	Function	n_{max}			
		128	256	512	1024
PETS09	Detection	49.763	47.958	48.812	48.609
	Likelihood	11.449	13.407	30.947	65.837
	Fusion	-	-	-	-
	Resampling	0.259	0.197	0.439	0.689
	Transition	0.709	0.905	2.108	3.734
	Remove	-	-	-	-
	Add	0.020	0.019	0.020	-
	UpdateTexture	-	-	-	-
PETS06	Detection	48.594	50.424	47.982	49.020
	Likelihood	11.253	16.723	28.515	52.913
	Fusion	-	-	-	-
	Resampling	0.347	0.514	0.179	0.585
	Transition	0.118	0.215	0.370	0.823
	Remove	-	-	-	-
	Add	0.010	0.004	0.005	-
	UpdateTexture	-	-	-	-
PETS03	Detection	48.980	64.055	63.962	63.515
	Likelihood	11.330	22.518	53.301	118.119
	Fusion	-	-	-	-
	Resampling	0.092	0.271	0.418	1.102
	Transition	0.613	2.346	5.338	7.760
	Remove	-	-	-	-
	Add	0.012	0.006	0.006	-
	UpdateTexture	-	-	-	-
EM330	Detection	32.947	33.510	32.652	33.129
	Likelihood	13.700	17.294	36.573	69.777
	Fusion	-	-	-	-
	Resampling	0.074	0.054	0.129	0.702
	Transition	0.036	0.054	0.223	0.713
	Remove	-	-	-	-
	Add	-	-	-	-
	UpdateTexture	-	-	-	-

TABLE 4.4: MOTA comparison with results in [153]

Author	MOTA	Appearance description	Estimator
Arsic09 [154]	22%	3D vertex model Detected volume in x,y and time	Detection[154] Data association (Normalized cuts [155])
Berclaz09 [156]	79%	3D visual hull 2D positions in ground plane	Detection (POM [101]) Data association (LP [156])
Breitenstein09 [157]	75%	Spatial pattern (HOG+ISM) 2D positions in image plane 2D positions in image plane	Detection Data association Bayesian (Particle filter)
Our method	93%	3D parametric model 3D parametric model+Texture	Detection to activate the tracker Bayesian (Particle filter)

4.5 Summary

Our detection and tracking algorithm is based on the parametric ellipsoid as described in Chapter 3. The parametric ellipsoid brings about a fast 3D likelihood computation method, which was never been studied in the multiple target tracking environment. A parametric model makes computation quicker as described in Section 3.3. The 3D ellipsoid model allows observation from multiple cameras to be integrated effectively. The advantage of a calibrated over a non calibrated system is accuracy shown in Table 4.4 (comparison between [101] and [157]).

In this chapter, we show that the MOTA of our tracking algorithm is higher than the best method in PETS09 workshop report [153]. This is based on our unofficial evaluation. In previous methods, detection was applied to transform the image observations to a set of detected positions in space before performing data association to link the existing trajectories with the detected position. This method is very common in multiple target tracking. However, our method uses detection to activate the particle filter and the particle filter uses a 3D model with an evolving texture signature to estimate the state variables. Thus, the state variable and texture signature are combined into a subject representation and eventually the state and signature together is used for likelihood calculation, unlike [157], which considers state variable and appearance separately. Integrating the texture signature into the likelihood function has the benefit that it can reduce the error rate as shown in Table 4.1. In Breitenstein’s study [157], he also used

classification to link detected subjects to trajectories but his work was based on 2D features, so it is difficult to extend the work from the single to multiple cameras. Using the ellipsoid texture learning method in our approach, the signature of all cameras can be defined consistently across all cameras. This results in higher MOTA as shown in Table 4.4.

We also studied the computational time of our framework for further parallel implementation. The parallel implementation divides the likelihood function into individual particles. This is an optimal solution because the number of targets (16) is always less than number of particles (512) . If we divide the likelihood function into individual pixels, we have to save output of individual pixels and then combine these outputs into a final likelihood value because of the nested structure of the for-loop in Algorithm 3.64. This will increase the total memory transfer compared to dividing by particle.

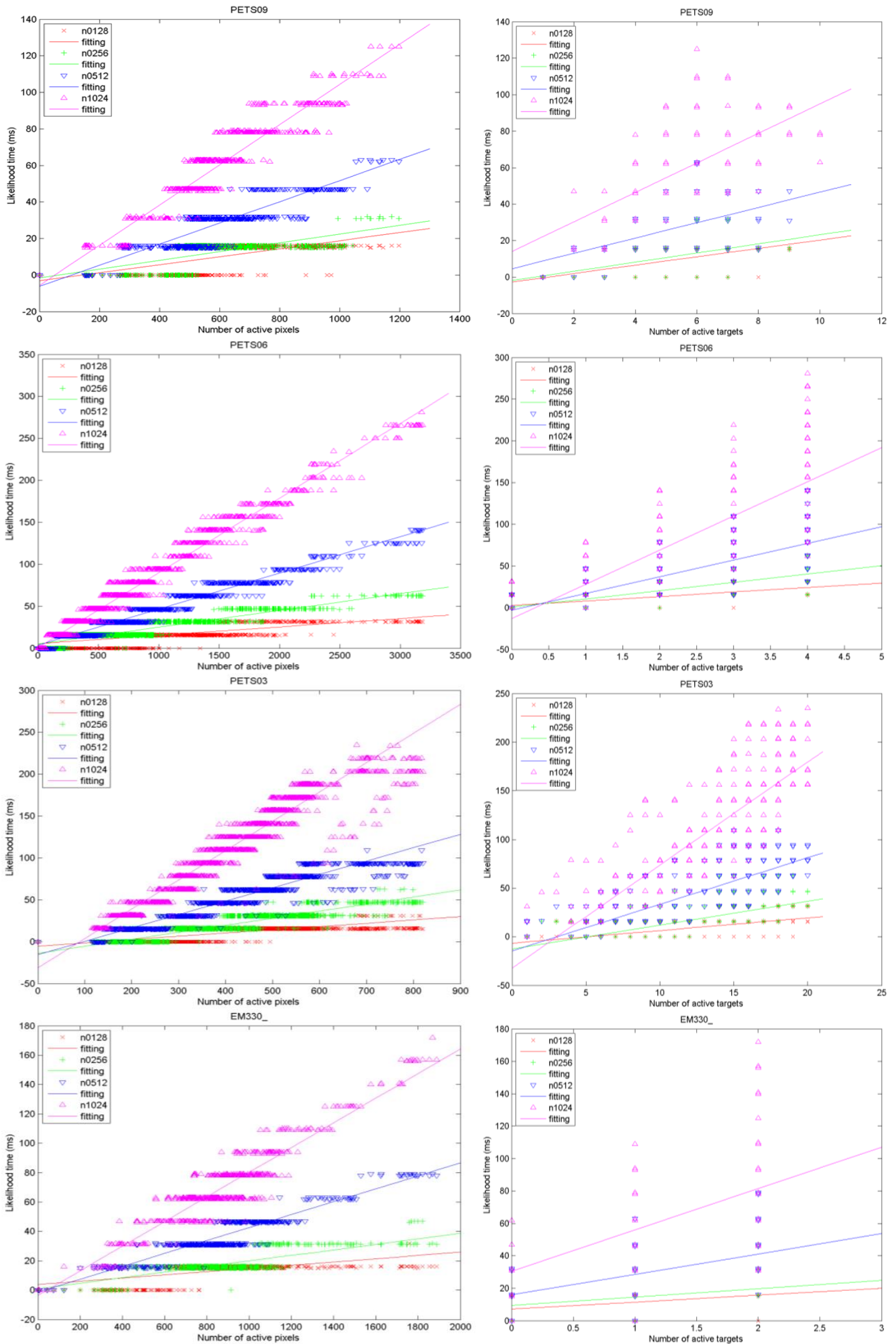


FIGURE 4.14: Likelihood Latency time.

Chapter 5

Parallel Implementation

In this chapter we discuss designing and implementing a parallel version of the tracking algorithm. In the particle filter, hundreds of particles express the probability density. Each particle state requires a complex likelihood calculation, which is similar to ray tracing in computer graphics. For a single subject, an area of the ellipse can be as large as several thousand pixels when the subject is very close to the camera. On average the number of calculations of ray-tracing in the likelihood computation is about 26 million, estimated from the average number of active pixels in an ellipse (10,000) times the average number of subjects (5) times the number of particles (512). Considering a frame rate of 7.5 fps, a number of floating point operations for each ray tracing is 105 floating operations (using ellipsoid projection $7.5s^{-1} \times 14\text{operations}$). Please see Table 3.1 for details of the estimation. On average, the likelihood function alone requires computational power just below 27GFLOPS (Giga Floating Point Operation Per Second). This is the number of operations for ray-tracing in the likelihood function only. From our estimation the whole system including detection and texture update requires up to 60 GFLOPS.

In Section 2.7 we gave a comparison and a justification for the target hardware platform. In this project we used a midrange hardware platform in order to build the prototype. The computational power from a specification of a standard CPU, for example an Intel Corei7 965 (4cores) can provide up to 69GFLOPS [10]. Our requirement is nearly reaching the theoretical maximum computational power of a standard CPU. A

midrange GPU such as Nvidia GeForce GTS250 can process 705 GFLOPS (128cores) [9], which is large enough for our tracking system requirement (60GFLOPS) at a normal frame rate of CCTV (7.5 fps). The GPU programming is also much easier and quicker compared to FPGA implementation as mentioned in Section 2.7. Due to the speed limitation and the difficulty of implementation as described earlier, we have decided to implement the tracking system on a GPU, which can deliver medium to high speed performance and take shorter time for development. Our design, implementation and testing is discussed in this chapter.

5.1 Parallel Design

For a worker a task can be performed in a sequential mode. When a number of worker increase the task can be split and performed in a parallel mode, where many sub-tasks are performed at the same time. The idea is very simple but in practice data transfer between the workers or processing cores has a limit. This limit makes the task splitting difficult and programmers have to concern about the different levels of memory architecture in the target hardware.

The maximum speed-up ratio of parallel processing comparing to a sequential processing can be predicted by Amdahl's law [169]. The maximum speed (S) is expressed in terms of the number of processors (N) and P a portion of code that can be parallelised compared to the total instructions of the code. Note that different instructions need different computational times so the portion P should be measured in units of time.

$$S = \frac{1}{(1 - P) + P/N} \quad (5.1)$$

For example suppose 80% of a serial instruction code can be parallelised. Another 20% cannot be divided *e.g.* memory transferring or sequential algorithm. Assuming all instructions use the same computational time then P will be 0.8. If we have 128 cores,

we will get a maximum speed-up ratio $S = 4.84$.

$$S = \frac{1}{(1 - 0.8 + 0.8/128)} \quad (5.2)$$

$$= \frac{1}{0.2 + 0.00625} \quad (5.3)$$

$$S = 4.84 \quad (5.4)$$

The fraction P/N is small when N is large, so the speed-up can be approximated by $S = 1/(1 - P)$.

5.1.1 GPU architecture

In this section we consider the processing units and memory architecture of GPUs. We have used the Nvidia GeForce GTS250. The Nvidia GPUs share a similar architecture, although they can have variations in the size of memory, the number of cores or even the number of bits of the FPU. The GTS250 is designed for processing single-precision floating point numbers or 32 bit floating point numbers. Hence, a variable in a GTS250 GPU program cannot be Double-Precision (64bit). More extensive models, which meet Nvidia CUDA compute compatibility of 1.2 [170] or higher, can process Double-Precision numbers. For this project we do not need to use Double-Precision numbers.

A GPU processor consists of many multi-processors (MPs) and each MP has 8 streaming processors (SP). The streaming processor is a special processor design for single instruction multiple-data (SIMD) parallel processing. In GPU programing the main computer is called *the host* and connected GPUs are called *devices*. Figure 5.1 shows the internal connections between different parts inside a GPU. The host CPU connects to the GPU via a controller or the global memory, so the CPU can access the instruction memory and the global memory but cannot access local memory, such as the shared memory, directly. In order to load local data in the MPs from the device to the host, the host has to send instructions to the device to perform a copy of the data from local memory to the global memory and then it can read data from the global memory. With the cooperation of the controller inside the GPU, the SP cores will be commanded to store/load the data to/from the global memory. The local memory, such as shared

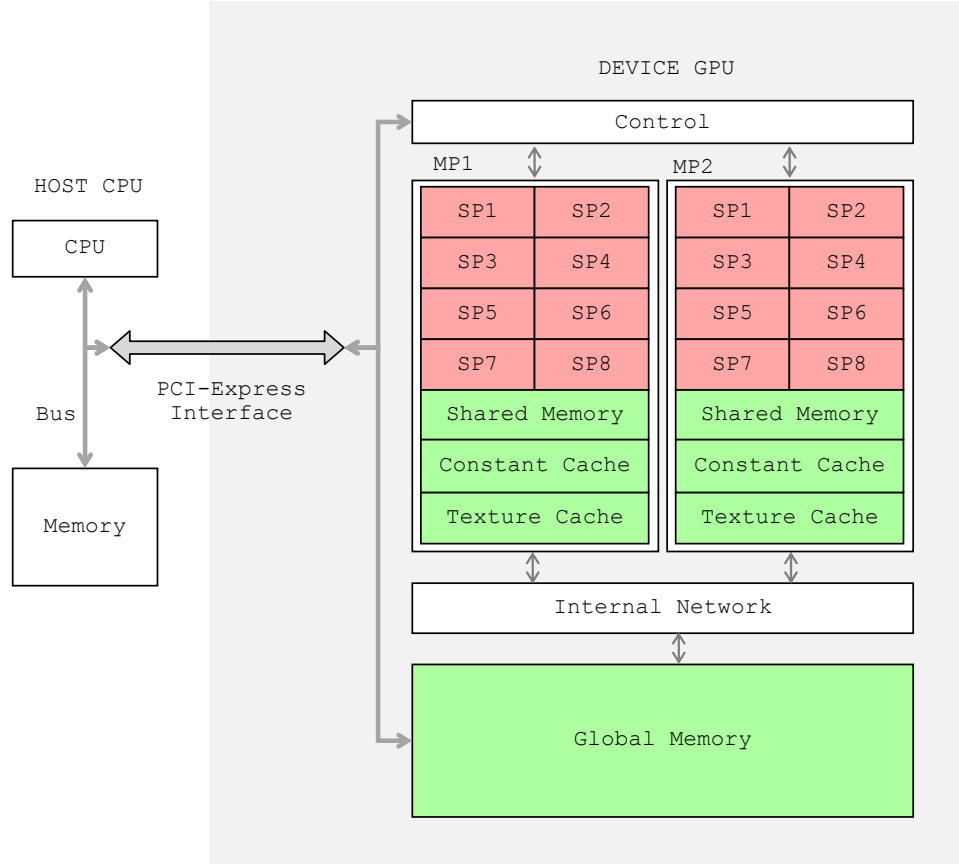


FIGURE 5.1: CPU to GPU connection and internal connection inside an example GPU (no practical GPU has 2 MPs).

memory and cached memory, are exploited in complex programming to improve the communication speed because the latency of accessing global memory from SP cores is huge: about hundreds of clock cycles, while, the latency to store/load shared memory is much faster a few clock cycles. Table 5.1 shows the accessibility of host and SP cores to read/write on different types of memory. The second column shows the size of the memory.

TABLE 5.1: Characteristic of device memory of GTS250.

Memory	Size	Cache size	Host access	SM access	Device Latency (clock cycles)
Global	1GB ²	-	R/W	R/W	100
Constant	64KB ¹	8KB ¹	W	R	1
Shared	16KB ¹	-	-	R/W	1
Texture	1GB ²	8KB ¹	W	R	1(hit)/100(miss)

¹From CUDA programming guide [170], in appendix G.

²From datasheet and confirmed by our testing.

Global memory The global memory is the largest memory. The global memory has flexibility to be read or written by the host or the device. One drawback of exploiting this memory in an application is its huge latency. Reading the global memory should be performed in batch mode called *coalesced access*. The smallest chunk of data that can be efficiently accessed by SPs is 64Byte for a device with compute compatibility 1.x (from section 3.2 in [171]).

Constant memory The constant memory can be written by the host but it is read only by SP cores. The constant memory has the ability to broadcast, which means all SP cores can read the same address in constant memory. The broadcasting allows all threads to read from same memory address. The broadcast ability is very useful when a program needs some parameters in many calculations, for example a physics constant in a simulation program or camera parameters in our work. Note that all threads must read the same address at the same time. If some threads read from different locations all threads will be serialised.

Shared memory The shared memory is located inside the MP unit so the SPs can access it in a few clock cycles. However, the size of shared memory is very small, so programmers have to design to apply this type of memory carefully. The shared memory is divided into equally sized memory modules (*banks*). Each bank can be accessed by a thread at a particular time. There are 16 banks in a device with compute compatibility 1.x [171], so, when many threads load the same memory bank the read conflict will cause serialisation. However, if 16 threads (a half warp of threads) read the same memory bank it will not conflict but the data will be broadcast to all 16 threads without serialisation. This fact is very important in order to use shared memory efficiently.

Texture memory The texture memory is a cached memory, which means that a SP core must search for needed data in local cache memory first before looking in the global memory. If it cannot find the data in the local cache (known as *cache-miss*), the request will be forwarded to the master controller to search in the global memory. If there is

a *cache-hit*, the read time will be shorter than reading from the global memory: about 100 times.

Register There is another types of memory with is called *a register*. The register is located inside every SP to store intermediate data after processing an instruction. The register can be read/written by the same thread, other threads in different SP cannot see this register.

CUDA programming Compute Unified Device Architecture or *CUDA* [170] is programming platform and software architecture for parallel processing developed by Nvidia. GPUs have been built in different configurations. In order to make pieces of code works properly in any GPU, Nvidia created the CUDA standard so a program can run on any CUDA-enabled GPUs. The standard is called *compute compatibility*. For example the GeForce GTS250, which is used in this project, has compute compatibility 1.1. We can check the compute compatibility from the specification datasheet before purchasing a GPU. There are hundreds of models of GPUs on the market and they have different numbers of SP cores. To make the same code able to run on different devices, CUDA introduced a logical structure of *grids* and *thread-blocks*. A grid is a collection of many thread-blocks. And a function to be executed by a grid is called *a kernel function*. Usually, processing in GPUs deals with huge amounts of data but with the same operation, so each SP has to repeat the same series of instruction many times on different data. A series of executed instructions is called *a thread*. The thread-block is a pack of threads in a grid. The size of the thread-block cab be varied from a few to hundreds.

At a particular time a SP unit processes a thread and a MP unit processes many threads in a thread-blocks as shown in Figure 5.2, where the vertical axis represents the direction of time. If we draw a horizontal line, the line will cross 16 threads being processed concurrently by 16 SPs of the hardware architecture in Figure 5.1. To complete all threads in the grid, the blocks must repeat 4 times to complete 64 threads. Figure 5.2 shows how 64 threads can be processed by 16 SPs. Considering the logical meaning of the thread-block and the grid, the number of threads per block (block-size) is 16 threads

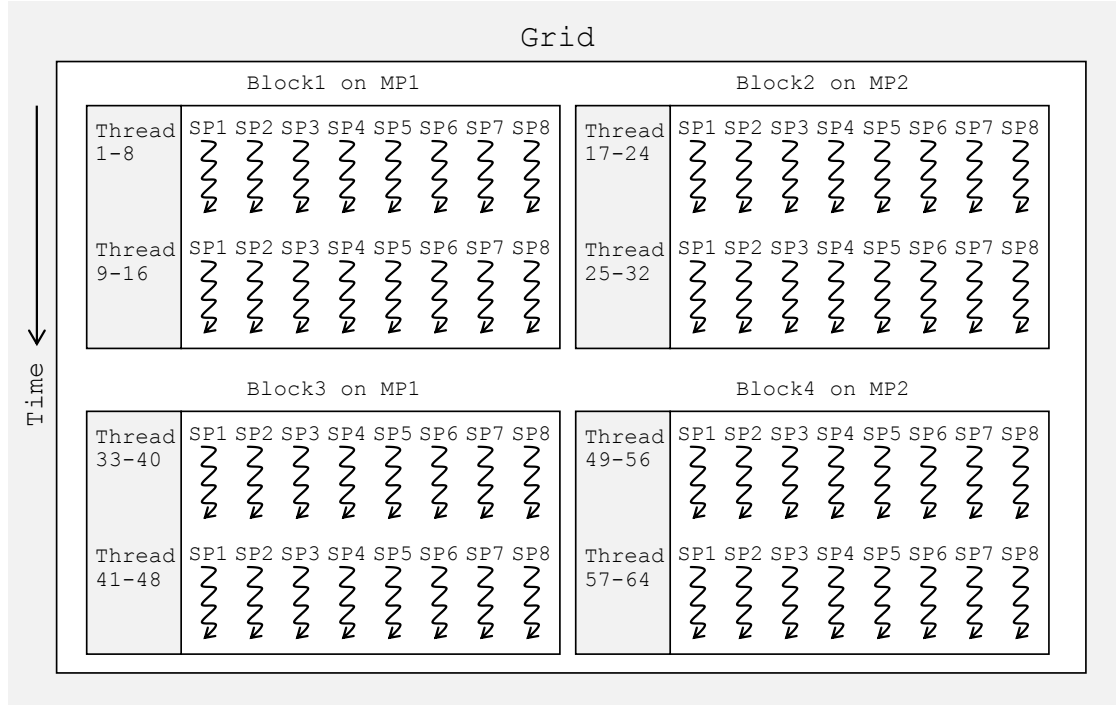


FIGURE 5.2: The logical thread-blocks are mapped to available 2 MPs.

as in Figure 5.2. One MP unit is always constructed from 8 SPs. This means a SP has to perform the process twice to complete a thread-block and a MP unit processes 2 thread-blocks to finish a grid of 64 threads, referring to the physical layout in Figure 5.1.

When the number of MPs increases, the time consumption decreases. For example, if we change the GPU to another one which has 4 MPs, which can process 32 threads at the same time and use half of the previous time. The logical structure of thread-block allows CUDA code to run on many variation of GPUs, with no need to indicate the number of MPs. In short, we do not need to know the number of cores on a target GPU before starting programming.

However, in order to optimize speed for a particular GPU, the best configuration strongly depends on the physical infrastructure.

5.1.2 Parallelism methods

There are many methods to implement algorithms in GPU. In this section, we discuss the four major techniques, which are applied in our tracking system; data parallelism, reduction, skip ahead and cache memory.

5.1.2.1 Data Parallelism

Michael Quinn classified two types of parallelism [172], *data-parallelism* and *control-parallelism*. Control-parallelism splits hardware into many parts and dedicates a partition of hardware to do a specific task from a set of several tasks. An example of control-parallelism is a task that consists of many independent subtask and many workers can be allocated for different sub-tasks. Normally different processors performs different operation on different data.

A special case of Control-parallelism is *pipelining*, where a series of instructions is divided into smaller pieces and a worker is allocated to perform a piece of task. To complete the whole task the data have to pass through all series of workers as similar as assembly line. At particular time there are many chunks of data being processed by those workers. This kind of parallelism is difficult to design. The total task must be divided equally to prevent any bottle-neck in particular step and to make sure the processing is finished before data is passed to the next worker. In high-level programing, processing time is difficult to control because it is managed by the operating-system so any interruption can cause delay and lead to unfinished false input to a successive worker.

Quinn also discussed data-parallelism, which means all workers do the same job on different inputs. Data-parallelism was introduced before Quinn by Michael Flynn in 1966 [173]. Flynn's taxonomy named data-parallelism as Single Instruction Multiple Data (SIMD) processing. Eventually, Single Instruction Multiple Threads or SIMT was described by Nvidia [170, 171]. The idea of SIMT is very similar to SIMD but SIMT has thread branching control. So the SIMT is an extension of SIMD with a thread controller. Data parallelism is simple to design and implement compared to the pipelining method because all workers do the same job and they finish at the same time. Data-parallelism is very useful when dealing with large numbers of inputs such as in image processing. For example, if we would like to compute subtraction between two images, we could pass the data of two pixels from the same coordinate of two images to an arithmetic core to compute and save the output in the global memory. Each thread processes an individual pixel coordinate independently. An example of CUDA programming on image subtraction is shown in the listing 5.1.

```

1 #define W 640    //width of the image
2 #define H 480    //height of the image
3 __global__ void kernelSubtract(int A[H][W], int B[H][W], int C[H][W]){
4     int x=blockIdx.x*blockDim.x + threadIdx.x;
5     int y=blockIdx.y*blockDim.y + threadIdx.y;
6     C[y][x]=A[y][x]-B[y][x]; //subtraction
7 }
8 int main(){
9     int* h_A, h_B, h_C;    //declaration pointers for host memory
10    int* g_A, g_B, g_C;    //pointers for global memory on the device
11    h_A = (int*) malloc (H*W*sizeof(int)); //host memory allocation
12    cudaMalloc(&g_A, H*W*sizeof(int));    //device memory allocation
13    ...
14    //more lines for reading images to h_A and h_B
15    ...
16    //copying h_A to g_A
17    cudaMemcpy( g_A, h_A, H*W*sizeof(int), cudaMemcpyHostToDevice);
18    //copying h_B to g_B
19    cudaMemcpy( g_B, h_B, H*W*sizeof(int), cudaMemcpyHostToDevice);
20    //define block size and grid size
21    dim3 BlockSize(16,16,1);
22    dim3 GridSize(W/dimBlock.x,H/dimBlock.y,1);
23    //proceed the kernel function
24    kernelSubtract<<<GridSize,BlockSize>>>>(g_A,g_B,g_C)
25    //copying g_C to h_C
26    cudaMemcpy(h_C,g_C,h*w*sizeof(int), cudaMemcpyDeviceToHost);
27    ...
28    //more lines for displaying the result
29    ...
30    //deallocate space in memory
31    free(h_A);
32    cudaFree(g_A);
33 }

```

LISTING 5.1: An example of data-parallel code in CUDA

CUDA is extended from the standard C language and it supports some object oriented functionalities in C++, such as polymorphism, default parameters, operator overloading, namespaces, function templates and classes (for device compute compatibility 2.0) [170]. Listing 5.1 shows the implementation of a data-parallel technique in the CUDA language. The main function starts at line 8. The code begins with the declaration of pointer variables for host and device. We normally use `h_` for indicating pointer of host and `g_` for pointers to global memory on a device. For variables `A`, `B` and `C` their pointers on the host and device are declared and allocated. For host pointers we can use `malloc(size)` to allocate memory, where the size is counted in Bytes. In this example, the pixel intensity is recorded in a 32 bit integer (4 Bytes) and the image has width of 640 pixels and 480 pixels in height. So the size of an image is $640 \times 480 \times 4$ Bytes. The memory allocation in a device can be made by using a CUDA function

`cudaMalloc(pptr, size)`, where `pptr` is a pointer of pointer to global memory. In line 17 and 19 host images of A and B are copied to the global memory on the device by `cudaMemcpy(target, source, size, direction)`. Then the dimension of thread-blocks and a grid are specified in lines 21 and 22. Note that the dimension of thread-blocks and a grid can be 3 dimensional. Next the kernel function is called and the dimensions of a grid and block are placed in order between the angle brackets `<<<GridSize,BlockSize>>>`. The device processes and stores the output in `g_C`. Finally, the output is copied back to the host. Note that the memory allocation lifetime is in the function scope. If allocations are made in the main function we must make sure that they are deallocated immediately after use, otherwise the system will run out of memory.

On the 3rd line, a new keyword `__global__` is added. The qualifier `__global__` expresses that the function is executed on a device and callable from the host only. Another type is `__device__` which is executed on a device and callable from the device only. The built-in variables `blockIdx` and `threadIdx` are indexes of the current thread-block and thread, respectively. The `blockDim` is the dimension of the block. From these three local variables; `blockIdx`, `threadIdx` and `blockDim`, the local thread can identify a global thread index effectively. So a local thread knows which pixel it has to process: in the example the pixel coordinate is determined by the indexes. In line 6, the input variables are read from the global memory and processed before the output `g_C` is written back to the global memory.

The Listing 5.1 gives an idea of how to implement a data-parallel process on a GPU. Most of the functions in the detection modules are processed by this data-parallel method, for example, background learning, foreground segmentation and image integration. The likelihood function is also implemented by this technique by splitting task according to the particle index so a thread computes the likelihood function for a particle state. Some variables are repeatably used in the likelihood calculation, such as camera parameters, ellipse parameters, transition parameters and grids. They are stored in constant memory for fast loading.

5.1.2.2 Reduction

In a large system consisting of many clusters of machines the tasks are mapped to each machine and the number of outputs is reduced to one. The idea of Map and Reduction was presented in [174, 175]. The Reduction method was applied in our frame work. Reduction is a computation to combine several inputs into an output, for example to find the maximum number from a large collection of numbers or computing the summation from many elements of data as in Figure 5.3. The Reduction technique is applied in many functions, such as, in the detection function to select the maximum response and in the re-sampling function for normalising the likelihood weights.

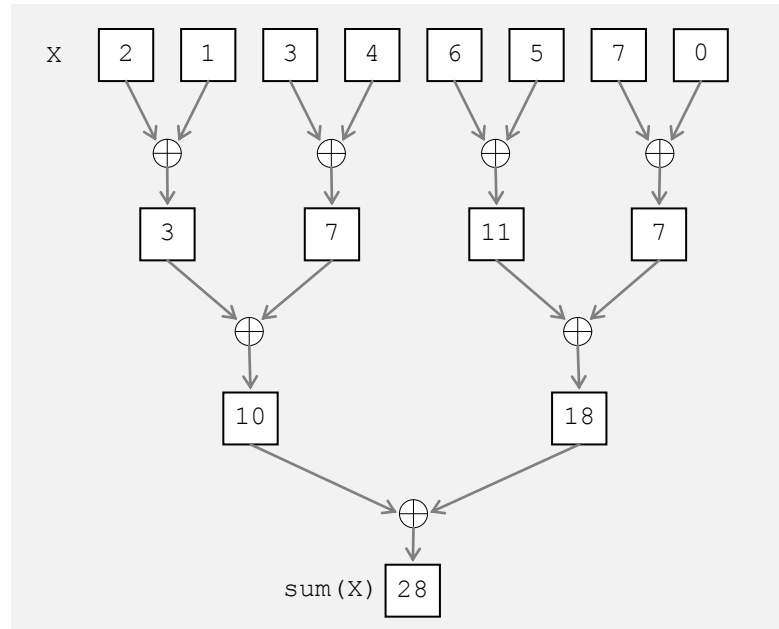


FIGURE 5.3: Compute summation by the Reduction method.

Mark Harris [176] suggested many possible techniques to optimise the Reduction method on a GPU. He considered the divergence of the control flow in a process and suggested methods to improve it by changing the read method from the shared memory. We applied his techniques, to find a maximum grid response in the detection function and to normalise the likelihood in re-sampling function. To summarise Harris's methods, he improved the Reduction process by;

- sequential-address data loading was the most effective method compared to the interleaved-address,

- processing and loading at the same time to reduce idle state during loading and
- decrease instruction overhead with the unroll for-loop.

The implementation involves 2 levels of Reduction. At the first level, many blocks load data from global memory and then compute outputs of their blocks and store the output on global memory. At the second level, a block loads all outputs of the first level to its shared memory before computing a final output and storing it back into global memory. It needs 2 levels because the shared memory cannot be seen from different blocks, so one way to connect them is to store the output of the first level in the global memory and then let a single block process the final result.

5.1.2.3 Skip ahead

The particle filter is derived from the Monte Carlo method and it generates samples by using a random number generator (RNG). The RNG algorithms were designed for a single processor by computing a series of numerical operators, such as low-discrepancy Sobol's sequences [128, 177]. The sequence was generated by applying a particular operator to a initial value and repeating to generate the sequence. The series of operations cannot be efficiently processed by a parallel processor directly. In data-parallelism, the task has to be divided into smaller pieces. To divide the process sequence, we have to look at the n^{th} step of the sequence. This is known as *skip ahead* or *jump*.

To achieve a good quality of uniform distribution with a low-discrepancy density in any volume of state space, a quasi-random generator such as the Sobol sequence [128] is exploited in the sampling method. The Sobol sequence can skip ahead to a particular step as described in [129]. In [178], Bratley showed that the skip ahead can be done by using the Grey code and a look up table of the direction numbers. The Sobol sequence is analytic and able to compute the value the particular n^{th} step. This makes the parallel implementation of skip ahead possible. The skip ahead method is very important in order to allow many threads generate uniform samples in a parallel mode. In the project, we used a CUDA function `curandGenerate()` in [129] to generate

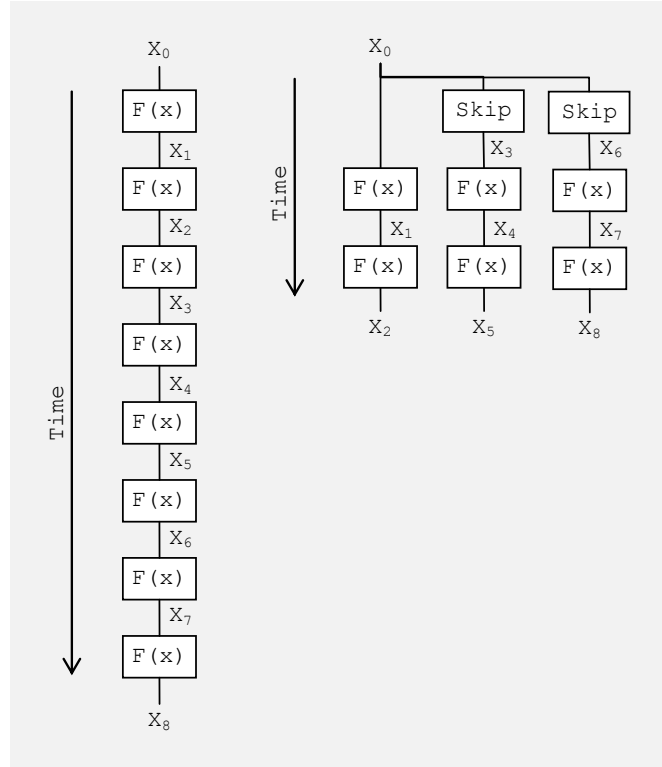


FIGURE 5.4: (left) a series of x_n are generated a sequential cascade process. (right) skip ahead is applied to divide the series into shorter sequences

samples in the transition function. The CURAND library has a skip-ahead function which is very useful in any parallel Monte Carlo simulation.

5.1.2.4 Use of fast memory

In order to minimise accessing time between cores and global memory, we used texture, constant or shared memory in many functions in the program. In this section we will explain where and how we applied the fast memory in the program.

In the detection module, the grid response function has thousands of grids and each grid has to compute the detection response. The computation is divided according to the number of the grid; each thread computes a grid response. So, there are many threads that read the foreground image at the same address, which causes read conflicts. So the foreground image must be stored in some fast memory, such as texture memory. We used texture memory to store the foreground image. A texture data array is located at

the same level of the global memory. The texture cache (the L1 cache of a MP) is bound to the texture array.

Moreover, the constant memory is used to store ellipse parameters of all subjects from previous iteration because the detection unit has to remove the present tracked subjects from the detection list to prevent the multiple-occupiers. All threads have to know the previous estimation result, which is expressed by ellipse parameters. The result is saved in constant memory, so all threads can load the ellipse parameters by the broadcasting method, which can prevent serialisation.

In the likelihood function, there are hundreds of particles and each particle is mapped to a thread. The thread reads data from the same pixel at a specific time period. Multiple reads from an address in the global memory can cause a read serialization (non-coalescing). The data need to be stored in cache. The texture memory was used in the likelihood function to store images. The likelihood function also uses a look-up-table in constant memory for fast calculation of the distraction penalising factor .

These methods are a few examples from the whole framework. In the implementation, the fast memory is also used for implementing other functions, which we cannot cover all of them. From the given examples, a reader can get an idea of how to use different types of memory in a specific algorithm. Further examples can be found in on-line sources or books, for example [170, 171, 179].

5.2 Implementation

In our experiment, the functions in the tracking framework have been parallelised, function-by-function, as described in the previous sections. Figure 5.5 shows the transfer of all functions from the host to the device and processing all functions on the device. Different algorithms in those functions perform best in sequential mode, whereas some functions can be improved to gain speed-up through parallelism. All functions have been implemented on the device to compare speed-up with the sequential algorithm in Figure 4.5.

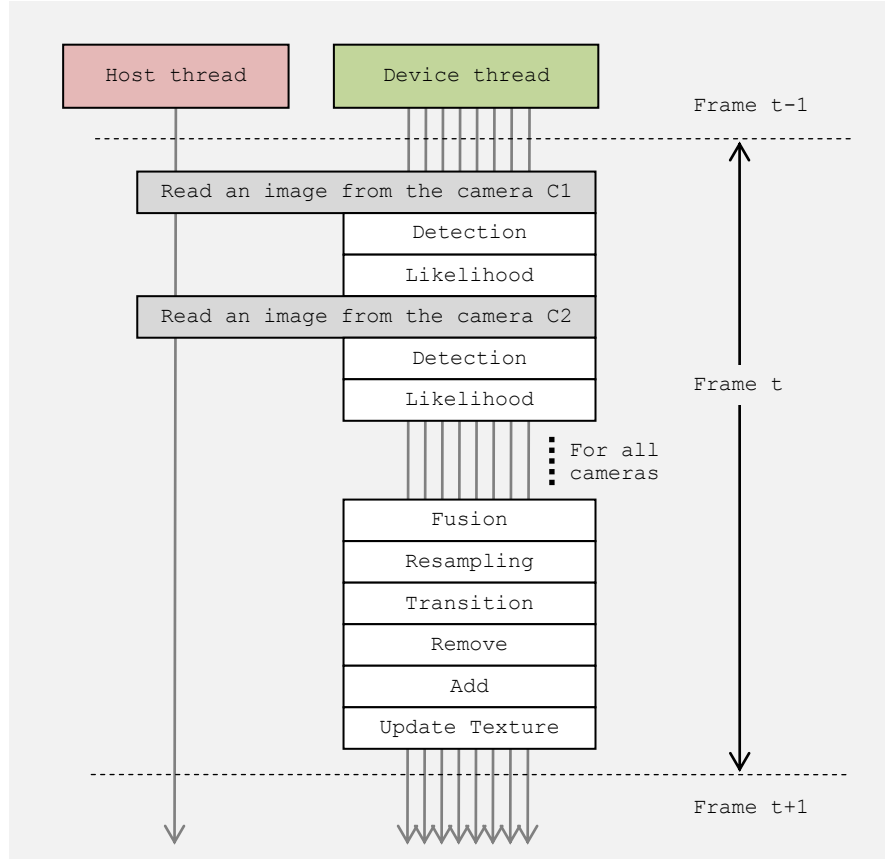


FIGURE 5.5: Sequential diagram of a GPU implementation.

We expected significant speed-up because the two slowest functions in our framework (detection and likelihood functions) can be parallelized by data-parallelism. Both of them involve many inputs, grids and particles, which are all independent. The detection function is divided into several grids and each grid is processed by a thread. Similarly, the likelihood function has many particles and each thread processes the computation of a particle. If the slow parts of the sequential implementation are parallelised that should bring substantial speed-up.

5.3 Speed evaluation

We tested the sequential and parallel algorithm using in two platforms; the CPU at 3.1GHz (used only one core) and the GPU GTS250, which has 128 of cores at clock frequency of 1.6GHz. The GPU was tested on the PETS09 dataset and we used *the CUDA profiler* to measure computation time and memory utilisation in every kernel function.

The CUDA profiler can measure useful information such as the utilisation of shared memory, registers, grid size, block-size and processor occupancy. This information is helpful in order to optimize computation speed.

Figure 5.6 shows a table of computation time for every function in the tracking program. The first and second columns show functions and kernel functions. A function can be constructed from many kernel functions, for example `detection` is constructed from 6 kernel functions and one CUDA memory-transfer function. The third column shows the GPU processing time per call of each kernel. The numbers in the third column exclude communication between the device and the host, such as loading instructions from the host and returning output to the host. The 4th column is the processing time per call including communication to the host. The 5th column is the number of calls of a kernel in the test, for example the likelihood function is executed 634 times. The 6th column is the percentage of computation time that the program spends on each kernel.

Figure 5.7 shows the block-size, grid size and memory utilisation of the kernels. The block-size is the number of threads in a block. The shared memory utilisation in the 5th column is the size in units of byte of the allocated shared memory for a block. Intermediate data is stored in registers during kernel execution. An MP has 8192 built-in registers and they have to be shared for all threads in a block. If the block-size is too large, a thread will have insufficient registers and the GPU has to use global memory instead, which makes it slower. The register utilisation per thread is shown in 6th column.

The *CUDA occupancy calculator* is a tool to help a programmer to optimise the speed of CUDA program. It needs three inputs from the CUDA profiler to identify a bottleneck and to determine a suitable block-size. The three inputs are;

- current block-size (the 4th column in Figure 5.7),
- size of allocated share memory per block (the 5th column in Figure 5.7) and
- the number of registers used by a thread (the 6th column in Figure 5.7).

Function	Kernel	GPU Time per call	Subtotal	Calls	%GPU time
Init 5.13ms	generate_seed	1.12	1.56	1	0.00
	Kernel_BG_Reset	3.31	3.33	2	0.04
	Kernel_PF_ResetAll	0.15	0.18	1	0.00
	Kernel_PF_ResetParticles	0.00	0.02	2	0.00
	Kernel_TXT_Init	0.01	0.02	1	0.00
	Kernel_TXT_ResetSign	0.00	0.02	2	0.00
Read 20.29ms	-		20.00		
	memcpyHtoD	0.03	0.29	1280	0.24
Detection 8.43ms	Kernel_Cam_gBGR2gRGBA	0.81	0.83	634	3.51
	Kernel_CAM_BG_Update	6.35	6.37	634	27.45
	memcpyDtoA	0.06	0.06	2533	0.98
	Kernel_IntRow	0.81	0.84	634	3.50
	Kernel_IntCol	0.24	0.26	634	1.05
	Kernel_Det_KernelInt	0.01	0.03	634	0.03
	Kernel_DET_RemoveFP	0.02	0.04	56	0.00
	Kernel_PF_Likelihood	13.57	13.60	634	58.69
Likelihood 13.6ms					
Fusion 0.09ms	Kernel_PF_DataFusion	0.04	0.09	634	0.18
Display 0.22ms	Kernel_PF_ShowParticles	0.06	0.10	317	0.13
	Kernel_RenderPBO	0.07	0.12	317	0.15
Add 0.31ms	Kernel_Max1	0.00	0.03	317	0.01
	Kernel_Max2	0.01	0.03	317	0.01
	Kernel_PF_Birth	0.01	0.02	3	0.00
	Kernel_Det_Reset_W	0.00	0.02	316	0.00
	Kernel_PF_Debug	0.17	0.19	316	0.36
	memcpyDtoD	0.00	0.02	1264	0.03
Resampling 0.56ms	Kernel_PF_Resampling	0.34	0.35	316	0.73
	Kernel_PF_ResetW	0.00	0.02	316	0.00
	Kernel_PF_GetExp	0.18	0.19	316	0.38
TexUpdate 0.30ms	Kernel_PF_MakeEllipsePar	0.01	0.03	632	0.04
	Kernel_TXT_Update	0.24	0.27	632	1.04
Display 0.21ms	Kernel_PF_ShowEllipse	0.06	0.08	28	0.01
	Kernel_TXT_ShowTxt	0.11	0.13	316	0.23
Transition 0.37ms	gen_sequenced	0.05	0.07	316	0.10
	Kernel_PF_MakeTransition	0.28	0.30	316	0.60
Remove 0.02ms	Kernel_PF_Punish	0.00	0.02	10077	0.11
Return 0.06ms	memcpyDtoH	0.02	0.06	2215	0.25

FIGURE 5.6: Computation time of all kernel functions.

Function	Kernel	%GPU time	block size	Share	Reg	Texture cache hit	Texture cache miss	warp serialize
Init 5.13ms	generate_seed	0.00	64	36	15	0	0	0
	Kernel_BG_Reset	0.04	256	24	5	0	0	0
	Kernel_PF_ResetAll	0.00	16	32	6	0	0	0
	Kernel_PF_ResetParticles	0.00	16	32	3	0	0	0
	Kernel_TXT_Init	0.00	256	28	5	0	0	0
	Kernel_TXT_ResetSign	0.00	256	24	5	0	0	0
Read 20.29ms	-							
	memcpyHtoD	0.24						
Detection 8.43ms	Kernel_Cam_gBGR2gRGBA	3.51	256	28	5	0	0	0
	Kernel_CAM_BG_Update	27.45	128	28	32	0	0	0
	memcpyDtoA	0.98						
	Kernel_IntRow	3.50	16	24	7	3.6E+06	1.2E+06	0
	Kernel_IntCol	1.05	16	32	6	0	0	0
	Kernel_Det_KernelInt	0.03	16	32	11	4.7E+03	2.7E+05	0
Likelihood 13.6ms	Kernel_DET_RemoveFP	0.00	16	60	24	0	0	1.3E+05
	Kernel_PF_Likelihood	58.69	16	32	30	1.2E+08	1.8E+07	3.0E+04
	Kernel_PF_DataFusion	0.18	16	36	7	0	0	1.1E+05
Fusion 0.09ms								
Display 0.22ms	Kernel_PF_ShowParticles	0.13	256	36	6	0	0	9.2E+05
	Kernel_RenderPBO	0.15	256	24	8	1.1E+06	1.1E+06	0
	Kernel_Max1	0.01	16	160	8	0	0	0
	Kernel_Max2	0.01	16	160	9	0	0	0
Add 0.31ms	Kernel_PF_Birth	0.00	16	40	16	0	0	0
	Kernel_Det_Reset_W	0.00	16	24	2	0	0	0
	Kernel_PF_Debug	0.36	16	32	6	0	0	0
	memcpyDtoD	0.03						
Resampling 0.56ms	Kernel_PF_Resampling	0.73	16	40	9	0	0	0
	Kernel_PF_ResetW	0.00	256	24	5	0	0	0
	Kernel_PF_GetExp	0.38	16	32	16	0	0	0
TexUpdate 0.30ms	Kernel_PF_MakeEllipsePar	0.04	16	28	24	0	0	0
	Kernel_TXT_Update	1.04	1	32	32	0	3.1E+02	0
Display 0.21ms	Kernel_PF_ShowEllipse	0.01	256	28	7	0	0	0
	Kernel_TXT_ShowTxt	0.23	256	28	6	0	0	0
Transition 0.37ms	gen_sequenced	0.10	64	48	22	0	0	0
	Kernel_PF_MakeTransition	0.60	16	36	20	0	0	0
Remove 0.02ms	Kernel_PF_Punish	0.11	1	40	2	0	0	0
Return 0.06ms	memcpyDtoH	0.25						

FIGURE 5.7: Memory utilisation of all kernel functions.

The inputs are entered in the CUDA occupancy calculator which draws three graphs of the relation between processor occupancy and the three inputs. Each graph will let the programmer know the bottle-neck of the kernel function. It also calculates a minimum number of blocks per MP for the current situation.

There are three types of possible situations that the calculator can identify, lack of blocks per MP, lack of registers or lack of shared memory. If the calculator indicates the first situation, lack of block per MP, it means the block-size is too big. The block size is large and unable to be allocated efficiently on each MP. Imagine we have 2 boxes of pencils and we need to count a total number of pencils. We have 10 workers able to count but the low number of boxes (packages) limits efficiency of the process. We can improve efficiency by reducing the size of the package (the block-size).

The second and the third situations are lack of shared memory or registers. We can solve this with two solutions. The first option is to minimize the use of intermediate variables in the kernel functions. Sometimes a kernel function must be split into smaller sub-functions. The second option is to reduce the block-size. When the block-size reduces, the number of active threads per MP decreases. We have to reduce activity because the physical resources are limited.

Our chosen device has compute compatibility 1.1 so it has been designed to compute efficiently at 24 warps per MP or 96 threads per SP, from Appendix G of [170]. When an MP processes 24 warps it shows that all SPs in a MP are busy or occupied by a task all the time. However, the occupancy does not indicate inefficiency of loading from the global memory or cache-miss. Loading must be considered individually by checking results from the profiler. Texture cache-hits and cache-misses are shown in column 7 and 8. The last column shows serialization due to memory bank conflict or control branching, which is caused by *control-flow*, *e.g.* `if-else`. The kernel function may consists of `if-else` control-flow. During executing some threads meets the `if` condition and some does not. Thus the master controller has to send individual instruction branch to SP instead of broadcasting the instruction to all SPs at once. It is called *instruction serialisation*.

From the table in Figure 5.6, the `Kernel_CAM_BG_UP` and `Kernel_PF_Likelihood` are the most complex as they consume more than 95% of the processing time. If we can

increase the speed of both processes, the overall computation time will reduce significantly. Table 5.2 compares the computation time between sequential and parallel implementations. There are improvements in Detection, Likelihood and Transition. The performance of Resampling and Add functions become worse than the CPU implementation, due to the sequential characteristics of those functions.

However, the processing times of these two functions are much smaller than the computation times of Detection and Likelihood. So, decreasing speed in those simple functions has an insignificant effect on the total speedup ratio.

Function	CPU time(ms)	GPU time(ms)	SpeedUp ratio
Detection	48.81	8.43	5.8
Likelihood	30.95	13.60	2.3
Fusion	-	0.09	-
Resampling	0.44	0.56	<u>0.8</u>
Transition	2.11	0.37	5.7
Remove	-	0.02	-
Add	0.02	0.31	<u>0.1</u>
TextureUpdate	-	0.30	-
Total ¹	82.3	23.2	3.5

TABLE 5.2: Comparing computation time between sequential and parallel processing.

5.4 Conclusion

We have discussed parallelising methods and implementation on a GPU. Due to the characteristics of the detection and particle filter that have many elements of input, these factors make increase in the overall speed-up ratio. We also optimised by exploiting fast memory and varying the block-size to obtain maximum speed.

We tested the parallel tracking system on the GPU, which has 128 cores at clock frequency 1.6 GHz. The sequential algorithm was tested on a single core of the CPU at 3.1GHz. Note that clock frequency of the CPU was about twice faster than the GPU. The detection speed is improved 5.8 times (or about 11 times, if the clock frequencies

¹processing time of Fusion, Remove and TextureUpdate functions are neglected

were the same) The Likelihood speedup is only 2.3 times due to serialisation and cache-miss problems. From the slower GPU we still improve the overall performance to yield 3.5 times speed-up.

The parallel implementation is faster than the sequential implementation but it can be faster than the current implementation. The parallel implementation needs improvement to get rid of serialization and cache-misses. In Figure 5.7 there is a lot of serialization (30,000 times) in the likelihood computation and there are many cache-misses. The serialization and cache-miss emphasise that there is a lot of room to further improve the GPU implementation.

Chapter 6

Tracking in a disjoint camera network

The tracking framework described in Chapter 3 and Chapter 4 is able to handle short-time occlusion by including the invisible state and the persistence level in the transition mechanism. The mechanism allows subjects to disappear for a few seconds (normally less than 2 seconds) but it cannot handle long periods of absence because this increases the uncertainty of estimation and increases the rate of distraction. Thus, the particle filter is unable to withstand a long period of disappearance. When a subject leaves or disappears for a long period the particle filter has to terminate the trajectory. The previous tracking framework is unable to identify re-appearing subjects, where there is a large gap between broken trajectories. In order to make an association between those broken trajectories we need a recognition system.

Figure 6.1 shows two configurations of camera networks. The joint field of view of camera network has an intersection between field of views, whereas the disjoint camera network has no connection between the camera views. In the disjoint network, the system loses observability when a subject moves between a camera node and neighbouring nodes.

Previously, the coverage of camera nodes had to be continuous in a joint camera network to prevent an unobservable region between camera views. However, a complete continuous network or a joint camera network is not always practical and limits

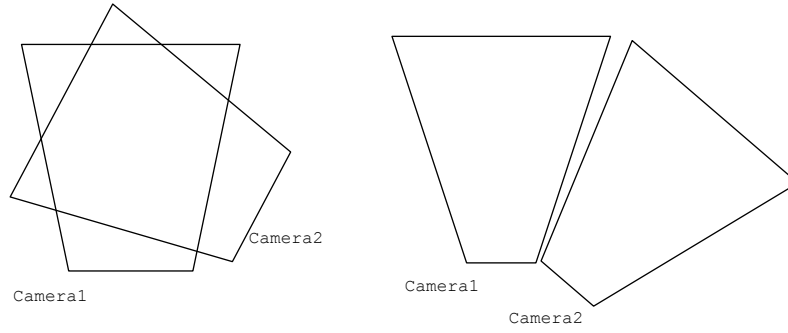


FIGURE 6.1: Coverage scalability of the joint and the disjoint network.

scalability. Obstacles and unobservable areas make a complete continuous configuration problematic. In the case of outdoor tracking, a disjoint network is more probable and free of installation constraints. However, tracking objects in a disjoint network is very challenging due to the unobservable space. The trajectory of a subject is broken into sub-trajectories and they must be linked together by data association or recognition.

A re-appearance event can occur in both joint and disjoint networks. Figure 6.2 shows trajectory transitions in a disjoint camera network. Subjects can re-appear in the same camera node or in other nodes. We tackle two main scenarios; re-appearance in a joint network and disjoint network. We are going to apply the Hungarian method [96] to these scenarios to connect sub-trajectories together by using state vectors and texture information. The matching score can be determined by spatial connectivity (a previously detected trajectory and a new detected location) and texture similarity. Our strategy is to exploit both spatial information and texture, which is acquired during the tracking process. The method will be tested on standard and our own datasets.

6.1 Related work

When subjects are able to move across FOVs (Fields of View), the tracking problem is more difficult compared to overlapping FOVs because of the unobservable gap. The non-overlapping camera network is also called the disjoint camera network. The challenges

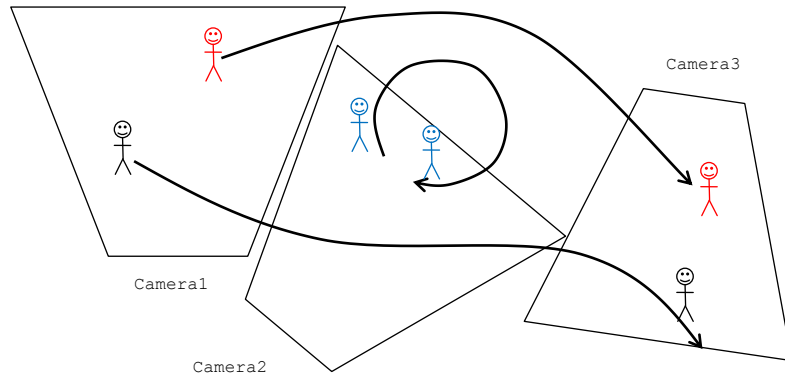


FIGURE 6.2: Transition of Subjects in a disjoint camera network

of tracking in a disjoint camera network are acquiring an invariant signature from a subject and assigning the correct ID to a subject.

View invariant appearance description is needed for recognition of the subjects. The appearance descriptor is always affected by the orientation of the cameras. The appearance descriptor of a subject may be totally different when it is captured from another camera. Imagine a subject who wears a red shirt and blue trousers. If the subject is captured by a camera from a side-view the colour histogram of the subject will consist of half red and half blue. When the camera is moved to capture the subject from a top-view, the colour histogram will be dominated the red colour. Thus, the colour histogram is not invariant signature.

An invariant signature, which has the same appearance in all cameras, is required in disjoint tracking. In [180], Kang proposed a 2D representation to transform the observation image to an invariant descriptor for disjoint camera network application. The 2D representation includes colour and edge encoded in 2D image representation. In [181], the 2D representation to express colour and spatial information of a subject was used as a signature. The new colour space (CI-DLBP) was introduced to make colour consistence in all cameras. The colour distribution on the 3D model, which is normally based on vertex base, is computational expensive and this make 3D representation never been used in tracking in disjoint camera network.

The geometry and topology of a camera network can be acquired by either training or proper calibration. The geometry or geographical information is very useful in order

to link a previous trajectory to a newly detected subject. The prior trajectory can also be applied to data association as in [182]. There, the topology network was modeled as a graph where the edges were travel time between FOV.

TABLE 6.1: Selected methods of disjoint camera tracking

Ref	Network topology	AD Signature	Tracking	Trajectory Association
Javed03 [182]	Inter-camera travel time	Colour histogram	MAP	HopcroftKarp [183]
Kang05 [180]	Calibration	Colour spatial encoding Edge spatial encoding	Kalman	JPDA
Madden07 [184]	none	Major Colour	Detection	Matching
LoPresti12 [185]	Inter-camera travel time node-to-node probability	Colour Histogram	Kalman	Dynamic Programming
Lian12 [181]	none	Colour spatial encoding Colour (CI-DLBP)	Detection	Matching

The appearance description signature is usually modeled by a histogram of colour *e.g.* [182, 185] or features on image plane *e.g.* [180, 181]. Table 6.1 shows that the colour is a popular choice for subject association. Other features such as edges are also used as a signature. The colour histogram differs from colour-spatial encoding in that that the colour-spatial encoding retains the spatial information of colour distribution on the image. Camera topology and the previous and the newly detected positions is also a key information to link broken trajectories to create a complete trajectory as described in [180, 182, 185].

6.2 Our approach

In traditional methods, the network topology can be modeled as a graph consisting of nodes and edges, where the edges represent inter-camera travel time or the probability of subject transferring from node-to-node. This covers more information compared the travel-time graph that the connection between FOVs is not single value as in the travel-time graph model. If the gap between between FOVs is a parallel gap the travel-time tends to be constant. However in practice, the gap is not necessary to be parallel gap, the distance between the edges of FOVs can vary. Therefore, the travel-time variation

depends on location of subject in the FOV. Unlike other work, we can predict the arrival time by using prior speed and position in an actual 3D coordinate system from the tracking described in Chapter 3.

The broken trajectories can be linked together based on signature only or on a combination of the signature and the prior state. The prior state contains position and velocity, which expresses subject motion on the ground plane. We call the state positions as *the trajectory information* to separate it from *the signature information*, where the latter is invariant in all cameras.

The published evaluation of the methods in Table 6.1 were produced from different datasets and metrics, which makes them difficult to be compared. In this work, we will test our method using the standard PETS09 and our dataset, which is available on-line.

We propose to use *the texture signature* described in Chapter 3. The subject signature we use here is the distribution of colour along the vertical direction of the ellipsoid surface. We do not use the azimuthal colour distribution because of uncertainty of the facing angle estimation. Hence, Our signature appearance description is invariant under orientation and transition transformation. We add the 3D geometry of subject and camera to the appearance model to make the signature appearance description consistent across all cameras. This allows us to link broken trajectories.

The colour feature is sensitive to illumination conditions. Therefore, we allow illumination changes by reducing the resolution of the colour-spatial encoding. The ellipsoidal surface is divided into 32 segments in the vertical direction (horizontal slices) and each segment has a RGB histogram, $16 \times 16 \times 16$ bins for RGB colour .

The trajectory spatial information is also considered as a key for recognition and identification. We use prior knowledge from the previous tracked trajectories and new detected trajectories. The trajectory data and the signature information are used for computing the association probability or *cost function* as described in Sections 6.3.2 and 6.3.3 in order to link many broken trajectories in a particular time. Note that there can be many trajectories and many new unknown subjects. The cost function is filtered by threshold before computing the optimum association by the Hungarian algorithm.

6.3 Theory

In single target tracking, an obvious presumption is that detections belong to the subject. However, in multiple target tracking, we must determine ownership between detections and subjects. Figure 6.3 shows the problem of association of the detection data with

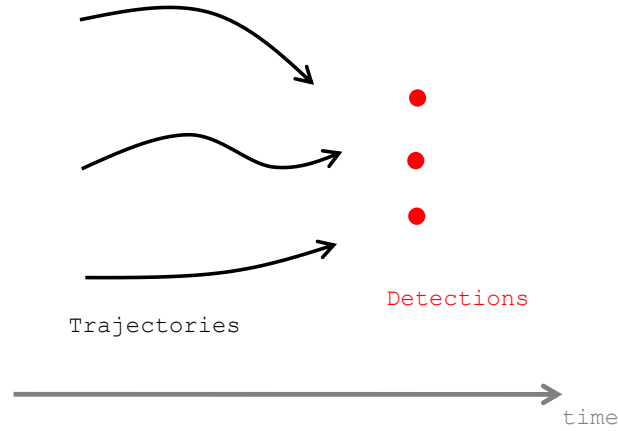


FIGURE 6.3: The data association problem; how do we link the detections with the trajectories?

the previous trajectories. In this case three detections and three trajectories can be associated using the shortest-path [82] algorithm or stochastic motion model [22]. Data association was used in multiple target tracking as described in Section 2.2.2.4, which was based on probability of ownership between a detection and a subject. The ownership probability defines a relation between a detection and a trajectory (an owner). In a particular time frame, many detections have to be linked with their trajectories and we can model the ownership probability. For example, JPDA [21] and PMHT [22] used only spatial probabilistic models to link detections (observation) to trajectories. Unlike previous methods of tracking by detection and association in [22, 82, 93], this study emphasises texture information, which is collectively acquired during the tracking process. The problem of finding an optimal solution to link many detections to many trajectories is similar to *the assignment problem*.

6.3.1 Assignment problem

The Hungarian algorithm [95, 96] has been designed to solve the *assignment problem*, which is to assign individual N workers to M separate tasks when those workers ask for different pay for different jobs. The Hungarian algorithm computes an optimal assignment that ensures all jobs are done within a minimum cost. The number of possible combinations of assignments is $\frac{M!}{(M-N)!}$, where $M \leq N$. So the optimal solution cannot be solved effectively by exhaustive search. The Hungarian algorithm was introduced in a logistics scenario by Kuhn [95] and it was revised and developed to a more general algorithm by Munkres [96]. In a particular time frame, the number of freshly-detected subjects is M and the number of known-missing subjects trajectory is denoted by N . Note that M and N are not necessarily equal.

In our method, the cost of matching a new detected subject to a previous trajectory is calculated from the similarity between the two textures and a spatial density diffusion function (a gating function). The association cannot happen if the similarity cost is lower than a threshold. This threshold enables a new unknown subject to be added to the known subject database.

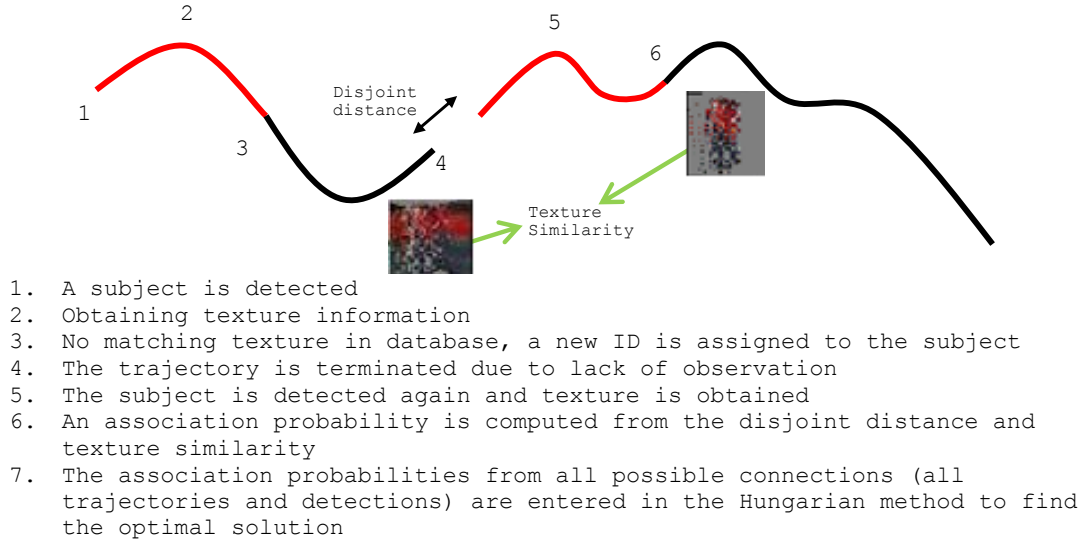


FIGURE 6.4: Texture matching in a video sequence, the trajectory line is broken at step 4.

Figure 6.4 shows our method of data association which is based on spatial and texture information. The red segments of the trajectories show when a new subject is detected

and the tracking system obtains a texture signature during tracking. The observed texture is accumulated over time to generate a histogram signature. Once the tracker fails to track, the histogram signature is stored in a database and checked against future detections. Histogram signatures of any new detected subjects will be checked against the previous known textures. When a histogram signature is found in a new trajectory, the two trajectories will be connected and the previous trajectory ID will be given to the detected subject.

6.3.2 Spatial association probability

The disjoint distance between the last position of the previous trajectory and a new detected position gives a clue to form an association. In order to associate the detections to the trajectories from the spatial information, we need to model the dynamics of the subject. We observe that people tend to move at the same velocity and have slight changes of velocity over time. We assume that human acceleration can be expressed by a simple Wiener process [159]. Equation (6.1) shows that the acceleration of a subject is a Wiener process (\mathcal{W}) with a control variance σ .

$$d\mathbf{Vel} = \mathbf{Accel}.dt + \sigma d\mathcal{W} \quad ; \mathbf{Accel} = 0 \quad (6.1)$$

$$\langle \mathbf{Vel} \rangle = \int \sigma d\mathcal{W} dt = 0 \quad (6.2)$$

$$\text{var} \langle \mathbf{Vel} \rangle = \langle [\mathbf{Vel} - \langle \mathbf{Vel} \rangle]^2 \rangle = \int \sigma^2 d\mathcal{W}^2 = \sigma^2 t \quad (6.3)$$

A stochastic variable consists of two parts; deterministic and stochastic parts. In Equation (6.1), the deterministic term is $\mathbf{Accel}.dt$ and the stochastic term is $\sigma d\mathcal{W}$. In this case, we set the expected value of the acceleration to be zero, $\mathbf{Accel} = 0$. This came from our assumption that people change velocity with minimum acceleration (force). If we use the Wiener process as to represent the stochastic part, the expectation of the variable will be the deterministic term. The variance will be the integration of the square of the stochastic part. According to the Wiener process characteristic, $d\mathcal{W}.dt = 0$ and $d\mathcal{W}.d\mathcal{W} = dt$. From Equation (6.2), the expectation of \mathbf{Vel} is zero, whilst the integration of square of the stochastic part will be $\sigma^2 dt^3$ as showed in and Equation (6.3).

We can determine the expectation and variance of the velocity as shown in Equation (6.4) and 6.5. Applying a similar method to position we get the variance as a function of $(\Delta t)^2$. The expected value, $\langle . \rangle$, represents the deterministic component, whereas the variance, $\text{var} \langle . \rangle$, expresses a boundary of randomness.

$$\langle \Delta \text{Vel} \rangle = 0 \quad (6.4)$$

$$\text{var} \langle \Delta \text{Vel} \rangle = \sigma^2(\Delta t) \quad (6.5)$$

$$\langle \Delta \text{Pos} \rangle = \text{Vel}_o \Delta t \quad (6.6)$$

$$\text{var} \langle \Delta \text{Pos} \rangle = \sigma^2(\Delta t)^2 \quad (6.7)$$

We assume that the variances of the acceleration in the x and y directions are equal (distributed as a 2D Wiener process). A 2D Wiener process is generated from a 2D Gaussian probability density function, (page 46) in [159]. This assumption keeps the expectation of velocity in same direction. The probability of detection occurring near the mean is high and reduced relative to the outward distance. There is a very low probability that a detected distance is beyond 2.5 times the standard deviation so a threshold on distance is considered $\text{distance} < 2.5\sigma(\Delta t)$. In order to reduce non-relevant observations, only non-assigned subjects in the circle will be tested by the data-association.

$$\eta^s(k, id) = \frac{1}{2\pi\sigma^2(\Delta t)^2} \exp\left(\frac{-r^2}{2\sigma^2(\Delta t)^2}\right) \quad (6.8)$$

$$r^2 = (\text{Posx}_k - \text{Posx}_{id})^2 + (\text{Posy}_k - \text{Posy}_{id})^2 \quad (6.9)$$

Equation (6.8) shows the spatial association probability between a detected subject index k and a trajectory index id .

6.3.3 Texture association probability

The tracking system is able to obtain a texture model from the surface of an ellipsoid by texture mapping and estimation as described in Sections 3.2.2 and 3.3.3. The texture signature consists of a 32-by-32 pixel array, where the row and the column of the texture

correspond to the vertical coordinate and the horizontal azimuth angle of the ellipsoid surface. Due to uncertainty of the facing angle estimation, the obtained texture is scattered in a horizontal direction. So we created a histogram of RGB colour from each row of the texture, keeping only the vertical spatial information. The histogram is accumulated during the tracking period. From each row of the 32-by-32 arrays of many time frames, we create a histogram in RGB colour space. A collection of histograms of all rows forms a *histogram signature*. This histogram signature is accumulated from the series of obtained textures. So after a short period of tracking the histogram signature is built up progressively as shown in Figure 6.4.

Once a subject is detected, the texture learning process starts immediately. The obtained texture will be checked against the histogram signature database by a Monte Carlo technique. Because a camera cannot see the entire surface of the subject at once, the pixels on the visible area are the available samples and they are tested by Monte Carlo estimation in Equations 6.10 and 6.11. Let $T_k(c)$ and $T_{id}(c)$ be histograms in colour space c of the detected subject (k) and the previous trajectory (id). Because $T_{id}(c)$ has small samples, the complete histogram cannot be constructed. In order to estimate the inner product between two histograms, the importance sampling method is applied. The integral of products between two density functions is estimated by the sampling method of Equation (6.11), where a texture data association cost function of each row z of the texture model is denoted by $\eta_z^t(k, id)$. Note that the texture coordinate system is (z, θ) as described in Section 3.2.2. The averaged texture cost function from all rows $\eta^t(k, id)$ (without subscript z) is computed and used as a texture association score.

$$\eta_z^t(k, id) = \int T_k(c, z).T_{id}(c, z)dc \quad (6.10)$$

$$\int T_k(c, z).T_{id}(c, z)dc = \frac{1}{imax} \sum_{i=1}^{imax} T_{id}(c_i, z) \quad ; c_i \sim T_{id}(c, z) \quad (6.11)$$

6.4 Design of Experiments

In this study the texture models are produced on-line and no texture model is created in advance. If the detection matches a trajectory texture model, the detected subject will be assigned with the trajectory *id*. Otherwise, the classifier must add a new category (a new subject) when there is no previous trajectory that matches a new detected subject. The new subject will be added directly to the database with a new *id*, when the texture is acquired with more than 30% of the total surface and has no match. Our classifier is based on threshold and maximum association score. In order to study the effect of spatial and texture scores, we consider these factors one by one in each experiment.

Recall rate To measure the recall rate, the assigned indexes (*id*) of all trajectories are monitored. The *id* of the same person of every trajectory is recorded. The number of correct connections is denoted by n_c , which is the number of correct links between the unidentified detected subject and the previous trajectory. The recall rate is calculated from the ratio $\text{recall} = n_c / n_{\text{total}}$, where n_{total} is a total number of time that a trajectory is split as shown in Figure 6.5.

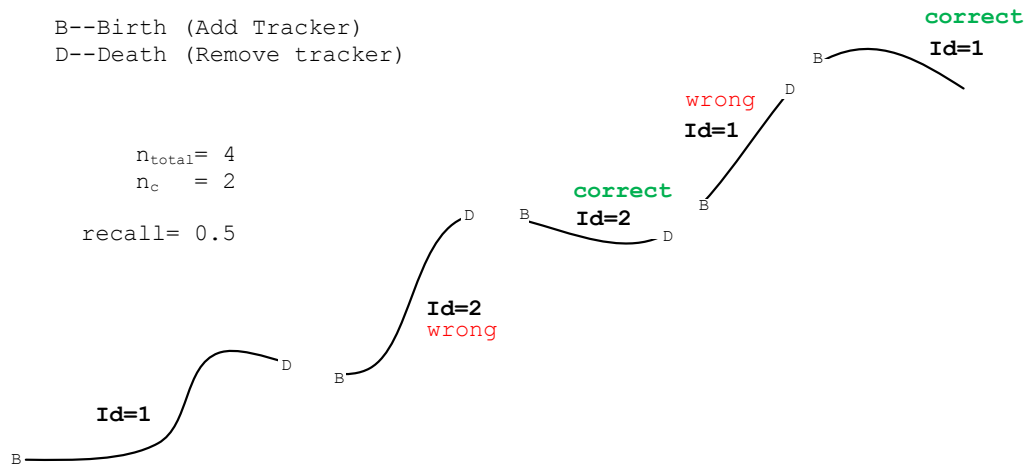


FIGURE 6.5: Recall rate evaluation.

6.4.1 Experiment 1: spatial-only

In this experiment, we used only the spatial association score $\eta^s(k, id)$ in order to link a detection to a trajectory. To compute the assignment problem, we generated the cost function $\eta^s(k, id)$ from Equation (6.8). For example, in a particular frame there are 3 subjects that are detected ($k = 1, 2, 3$) and 2 trajectories that have no trackers associated ($id = 1, 2$), so we can generate a cost matrix \mathbf{F} from the $\eta^s(k, id)$ as shown in Equation (6.12).

$$\mathbf{F} = \begin{bmatrix} F_{1,1} & F_{1,2} \\ F_{2,1} & F_{2,2} \\ F_{3,1} & F_{3,2} \end{bmatrix} \quad (6.12)$$

where

$$F_{k,id}^s = \text{ramp}[\eta^s(k, id) - \nu^s] \quad (6.13)$$

The element of the cost matrix is a cost function $F_{k,id}^s$ (the super script “s” denotes the spatial information) is computed from the *ramp function* [186] with a threshold ν^s . The ramp function is linear when the input is positive and zero when the input is negative.

$$\text{ramp}(x) = \begin{cases} x & ; 0 < x \\ 0 & ; \text{otherwise} \end{cases} \quad (6.14)$$

After the matrix \mathbf{F} is computed, we use the Hungarian method as described in Section 6.3.1 to find the optimal solution. In this case, the optimal solution is a configuration that maximises the total cost. In this experiment we vary the threshold ν^s from 1×10^{-3} to 1×10^{-2} and also alter the value of σ in Equation (6.8) between 0.6 and 1.4 m/s^2 . Then we measure the recall rate as described in Section 6.4.

6.4.2 Experiment 2: texture-only

Next, we repeat the experiment as described in Section 6.4.1 but this time we consider only the texture association score. The cost is computed from Equation (6.15).

$$F_{k,id}^t = \text{ramp} [\eta^t(k, id) - \nu^t] \quad (6.15)$$

In this experiment we vary the texture threshold ν^t from 5×10^{-3} to 5×10^{-2} and measure the recall rate.

6.4.3 Experiment 3: combining spatial and texture

Again we repeat the procedure in Section 6.4.1. The combined cost function is computed from the product of the spatial and texture cost functions.

$$F_{k,id}^{st} = \text{ramp} \left[(\eta^s(k, id))^\xi \times \eta^t(k, id) - \nu^{st} \right] \quad (6.16)$$

We choose the best value of σ from previous experiments. The factor ξ is a combining factor to adjust importance between spatial and texture terms. We vary ν^{st} from 10^{-9} to 1 and ξ between 0.0 and 5.0. We would expect that the combination of cost function improves the recall rate compared to the previous experiments that use single cost function.

6.5 Evaluation

The data association method is evaluated using the standard PETS09 dataset and our own dataset Dec11 (we explain the Dec11 dataset production method in Appendix B). After obtaining the dataset we followed the evaluation method described in Section 6.5.1. The experiment includes the effect of tuning parameters on recall rate.

6.5.1 Evaluation by using PETS09

We consider a joint camera view network where people in the scene can move in and out across the observable region. As stated, the previous tracking framework cannot handle long periods of disappearance of subjects followed by re-appearance events.

In this experiment the broken trajectories were linked by the data association using spatial and texture information as described in Section 6.4. We used image sequences from the 1st and 3rd cameras of the PETS09 dataset to evaluate our data association method by measuring a recall rate as described in Section 6.4.

Because the PETS dataset is fairly short, 795 frames or 106 seconds, it has a small number of reappearance events. Therefore we extended the sequence by processing forwards and backwards 4 times, from the first to the last frame, then backwards to the first frame. This method resulted in a jerking motion at the first and last frames. This is unrealistic and makes the trackers lose the subjects, breaking the trajectory and affecting the tracker. However, the number of trajectory splits during the sequence is far more than the number of splits due to the sudden motion change in the first and last frame. In addition, the forwards-backwards testing affects the tracker but should not affect the recall rate. The evaluation method consists of three steps, tracking, data association and measuring the recall rate.

Tracking The first step is to perform tracking by using the ellipsoid model. The output of state vectors and texture models of all active subject was stored in files. The history of the trackers was stored in an xml file and processed by the data association module.

Data Association The data association from Section 6.3 has been implemented in Matlab. The data association computation included the spatial and texture cost functions described in Section 6.3. The history of state vectors was used to generate trajectories and the texture models were accumulated to produce the histogram signature as discussed in Section 6.3.3. For each experiment, the output was saved in both image and text formats for further investigation.

Measure recall rate The recall rate was evaluated by the method in Section 6.4. Note that we had many trajectories, so the number of correct assignments n_c was counted from all trajectories.

Output from data association Two frames of sample results are shown in Figure 6.6. The colour code and numbers in rectangular boxes represent the signature indexes (id). The right panel shows the texture and connections between the current trackers (left column) and the trajectory signatures (right column). The numbers on the vertical axis are the tracker index k and signature index id , respectively. The histogram signatures were accumulations of observations that were provided by the trackers. The top-left panel represents the positions of subjects in units of a metre and the size of a circle is the gating function. Any unknown subjects appearing in the gating circle were tested and matched by spatial-texture data association. The black dot shows the direction of velocity of the tracked subject. In the 646th frame, a new subject is represented by black small circles at a coordinate around $(-8,0)$. In the next frame, the texture of the subject was found and matched to $id=2$ (dark green colour). The images below on the left show the re-projection of the 1st and 3rd cameras. The static occlusion due to the tree can be solved by integrating multiple camera.

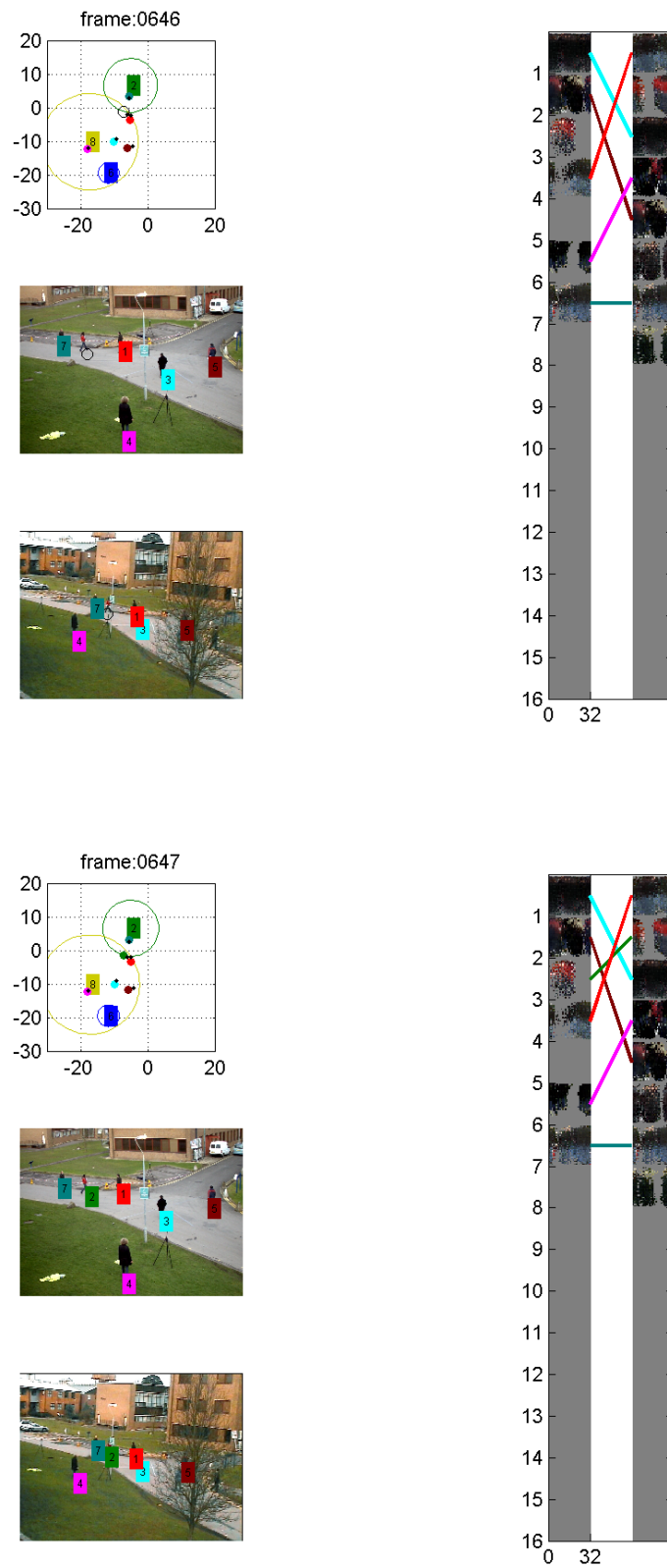


FIGURE 6.6: Spatial-texture data association with PETS09 dataset, detail in text

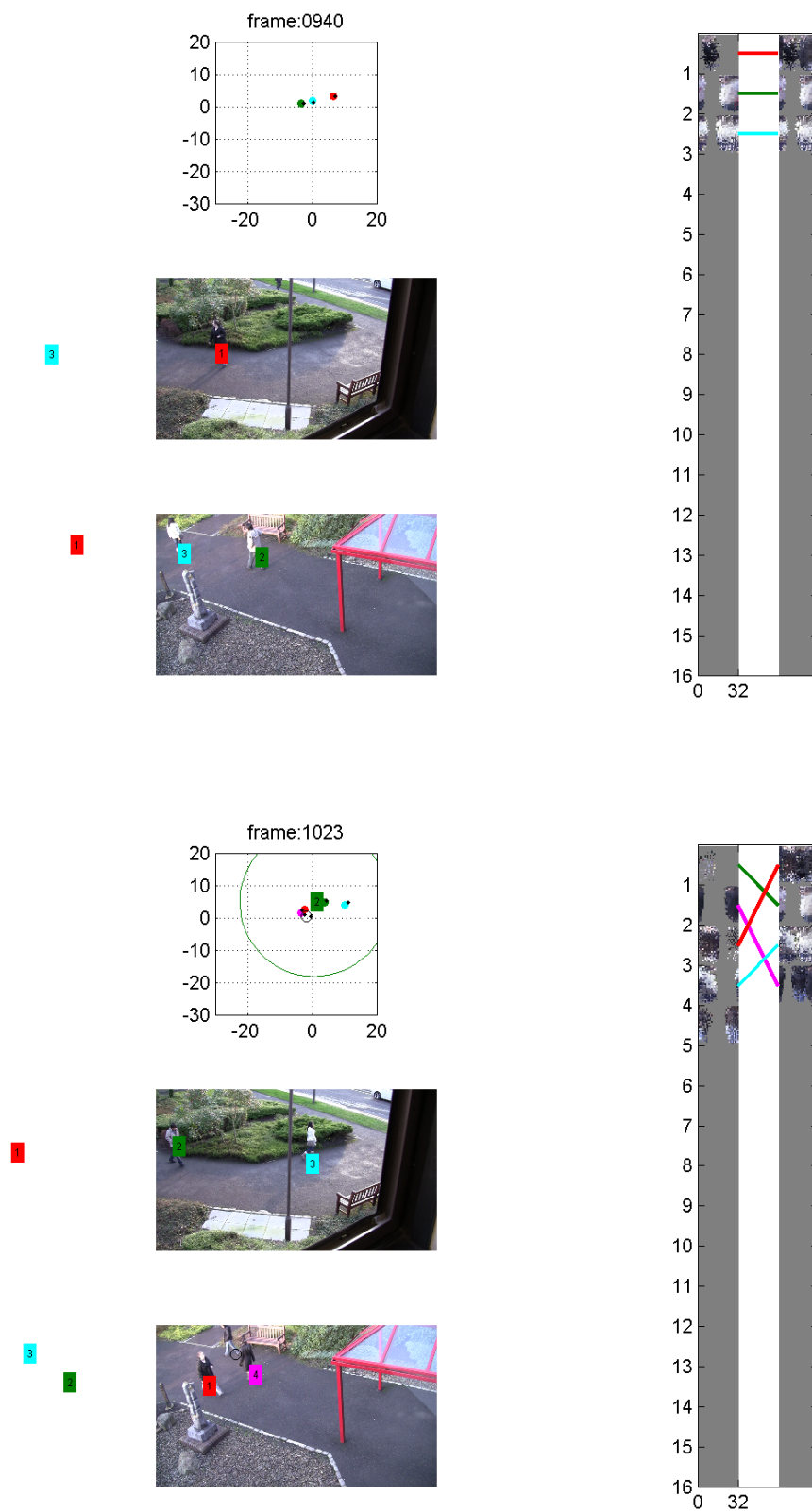


FIGURE 6.7: Data association with DEC11 dataset. These samples are taken from the *experiment2* with $\nu^t = 2 \times 10^{-2}$

6.5.2 Evaluation by using DEC11 dataset

We repeated the experiments but this time we changed the dataset from PETS09 to DEC11 (our own dataset). The procedure to generate the DEC11 dataset was described in Appendix B.

6.6 Result

In PETS09 evaluation, we considered all subjects as in Figure 6.8 and measured the recall rate by comparing the ground truth trajectory and assigned id of each sub-trajectory. The series of result images is similar to those of Figure 6.6. We noted the frame index and ID of individual subjects when the subject re-appeared again we compared the assigned ID with the previous ID. If the ID was assigned correctly, the counter n_c was increased. The same recall method was used in DEC11. The subjects and captured screen images are shown in Figure 6.9 and 6.7.



FIGURE 6.8: Subjects and their labels in the PETS09 sequence.



FIGURE 6.9: Subjects and their labels in the DEC11 sequence.

6.6.1 Result 1: spatial-only data association

Table 6.2 and 6.3 show the results of *Experiment 1* in Section 6.4.1. The table shows the recall rate with varying threshold ν^s and variance σ . In PETS09, the best parameters are $\nu = 3 \times 10^{-3}$ and $\sigma = 0.8[m/s^2]$, which yield the recall rate of 62%. The maximum correct recall rate in DEC11 is 64% with $\nu = 1 \times 10^{-3}$ and $\sigma = 1.0[m/s^2]$

TABLE 6.2: Recall rate of the spatial-only data association (PETS09)

$\nu^s [\times 10^{-3}]$	$\sigma[m/s^2]$				
	0.6	0.8	1.0	1.2	1.4
1	38	54	62	54	46
2	46	46	46	62	62
3	54	62	54	62	54
4	54	62	62	54	54
5	46	54	38	38	46
6	54	54	38	46	46
7	46	54	54	54	46
8	46	54	54	54	46
9	46	54	54	54	62
10	46	54	54	54	62

TABLE 6.3: Recall rate of the spatial-only data association (DEC11)

$\nu^s [\times 10^{-3}]$	$\sigma[m/s^2]$				
	0.6	0.8	1.0	1.2	1.4
1	57	50	64	57	43
2	57	43	50	43	36
3	43	36	43	36	29
4	43	36	43	36	14
5	29	36	36	21	14
6	29	36	36	14	14
7	21	36	14	14	14
8	21	29	14	14	14
9	21	14	14	14	14
10	21	14	14	14	14

6.6.2 Result 2: texture-only data association

In *Experiment 2*, we used only the texture cost function as described in Section 6.4.2. Table 6.4 shows the recall rate measured from the datasets. The recognition method worked more accurately in DEC11, which had a 79% correct recall rate.

TABLE 6.4: Recall rate of the texture-only data association (PETS09 and DEC11)

$\nu^t [\times 10^{-3}]$	PETS09	DEC11
5	23	43
10	62	29
15	62	57
20	54	79
25	46	79
30	46	79
35	46	79
40	23	79
45	31	79
50	31	43

6.6.3 Result 3: Spatial-texture data association

The cost function in Section 6.4.3 was tested and the results are shown in Table 6.5 and 6.6. We believed there should be a higher recall rate from the combined cost function but we did not see the peak in the second table. Thus, we decided to redo the experiment with smaller increments of ξ and ν^{st} . The results are shown in Figure 6.10 and 6.11. In PETS09, the recall rate was highest at 84.6% with $\log_{10}(\nu^{st}) = -3.7$ and $\xi = 0.6$. This result shows that the combined cost function gives a better result compared to separate cost functions. In case of DEC11, the peak was at 71.4% with $\log_{10}(\nu^{st}) = -1.7$ and $\xi = 0.0$. However, the combining factor $\xi = 0$ means it is best to exclude spatial from the cost function. The bigger ξ , the more emphasis on the spatial data as shown in Equation (6.16).

FIGURE 6.10: Results of the spatial-texture data association PETS09

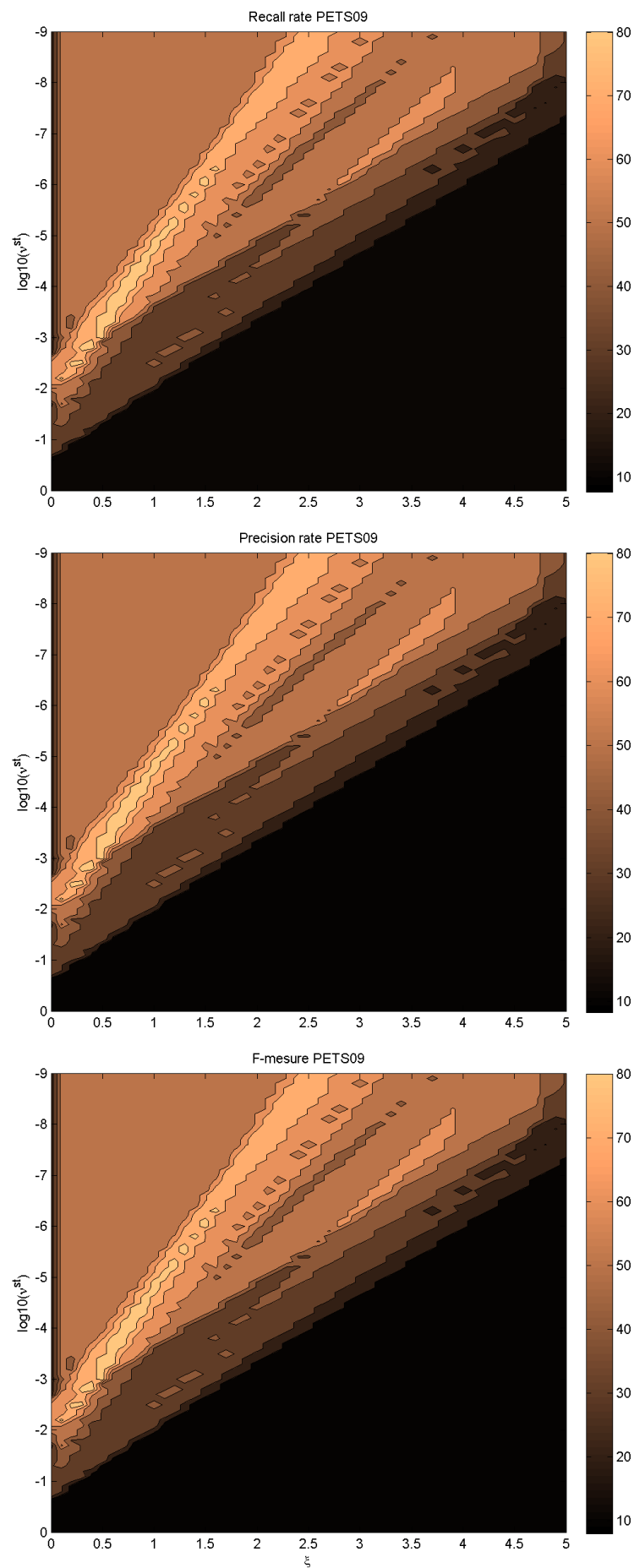


FIGURE 6.11: Recall rate of the spatial-texture data association DEC11

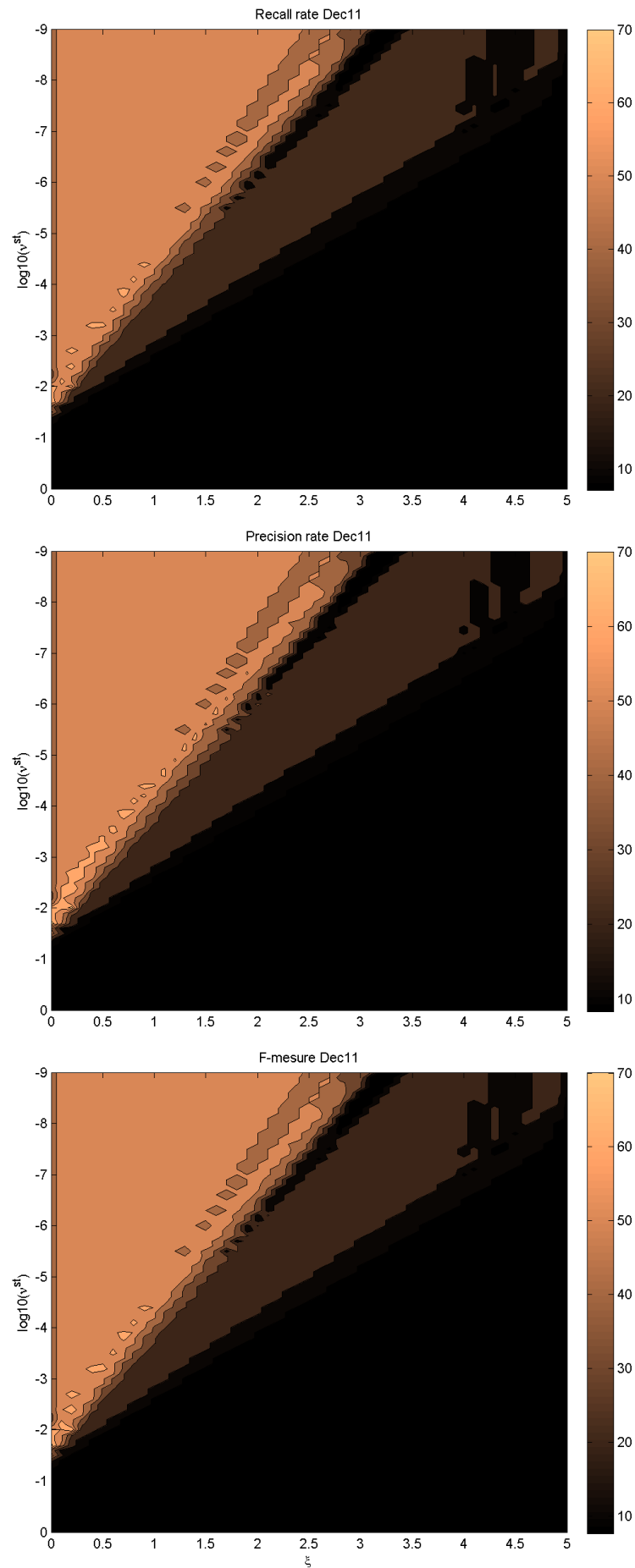


TABLE 6.5: Recall rate of the 2nd combining method (PETS09)

$\nu^{st} [\times 10^{-8}]$	ξ				
	1.0	2.0	3.0	4.0	5.0
1	54	62	54	54	38
2	54	77	46	54	54
3	54	77	54	54	31
4	54	77	54	46	31
5	54	77	54	46	23
6	54	69	54	46	23
7	54	69	54	46	23
8	54	69	38	46	23
9	54	69	54	46	23
10	54	69	54	46	23

TABLE 6.6: Recall rate of the 2nd combining method (DEC11)

$\nu^{st} [\times 10^{-8}]$	ξ				
	1.0	2.0	3.0	4.0	5.0
1	57	57	21	21	14
2	57	50	21	21	7
3	57	43	21	14	7
4	57	43	21	14	7
5	64	43	21	21	7
6	57	50	21	21	7
7	57	50	21	21	7
8	57	50	21	14	7
9	57	50	21	7	7
10	57	50	21	7	7

6.7 Recall Precision and F-measure

Figures 6.10 and 6.11 show the recall, precision and F-measure of the experiments. These measurements were also described in [187]. The recall is computed from the number of correct assignments estimated by our algorithm ($tp = nc$) and the total number of assignments of ground-truth ($tp + fn = n_{total}$) as showed in Equation (6.17). The precision rate is a ratio between the number of correct assignments (tp) from the algorithm divided by the number of total ($tp + fp$) assignments, where tp , fp and fn

are true positive, false positive and false negative.

$$\text{Recall} = \frac{tp}{tp + fn} \quad (6.17)$$

$$\text{Precision} = \frac{tp}{tp + fp} \quad (6.18)$$

$$F = \frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}} \quad (6.19)$$

According to Figure 6.5, $\text{Recall} = n_c/n_{total}$. The n_c is the number of correct assignments and n_{total} is the total number of assignments. The recall has the same definition.

The patterns of recall, precision and F-measure are very similar because the tracking system produces low miss rate (about 4% per frame) as described in Table 4.2. Thus, the chance of a tracker missing the subject per sub-trajectory is very close to zero (a product of miss rate per frame). This leads to nearly zero false negatives (fn) due to miss detection. The only reason for a rise in fp and fn is faulty assignments, and those assignments must be bijection (one-to-one correspondence) because of the Hungarian algorithm constraint. Therefore, $fn \approx fp$ and $\text{recall} \approx \text{precision}$ and this makes recall, precision and F-measure very similar.

We also plot the ROC graph to compare our classification data association with other method such as [184]. However, the ROC graph is linear with a negative slope as showed in Figure 6.12, which is unusual and it is resulted from the fundamental characteristic of the data association algorithm.

Figure 6.13 shows association of a trajectory (**A'**) with 3 unknown subjects (**A**, **B** and **C**). The correct association is a connection between the subject **A** and the trajectory **A'**. For each trajectory the system can make a correct (left diagram) or wrong (right diagram) connection. If it makes a correct connection $tp = 1$, otherwise $fn = 1$. So, we can write Equation (6.20) for each trajectory.

$$tp + fn = 1 \quad (6.20)$$

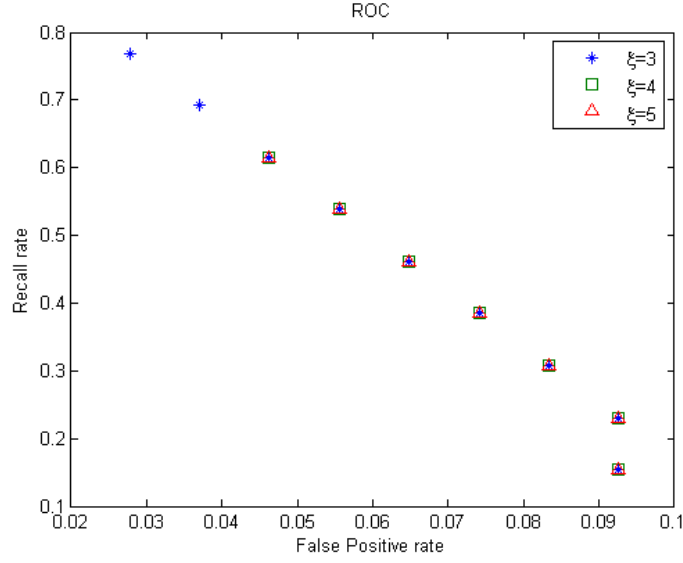


FIGURE 6.12: ROC computed from PETS09.

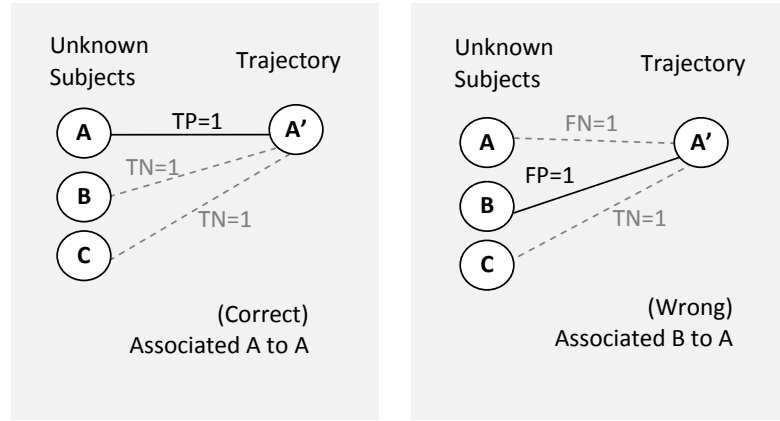


FIGURE 6.13: Characteristic of our data association.

Once the system makes a wrong connection it immediately generates both fp and fn .

$$fp = fn \quad (6.21)$$

The ROC plot is generated from the FPR(false positive rate) and the recall rate. The FPR is computed from Equation (6.22).

$$FPR = \frac{fp}{fp + tn} \quad (6.22)$$

As you can see the denominators of Equations (6.22) and (6.17) are constants. Substitute Equation (6.21) in Equation (6.20) and use the fact that the denominators of

Equation (6.22) and 6.17 are constant numbers. We can write Equation (6.25). P and N are the numbers of positives and negatives, which are constants in every dataset.

$$tp + fp = 1 \quad (6.23)$$

$$\text{Recall} \cdot P + \text{FPR} \cdot N = 1 \quad (6.24)$$

$$\text{Recall} = 1 - \frac{N}{P} \text{FPR} \quad (6.25)$$

In conclusion, the ROC plot is not suitable for data association evaluation. Furthermore, our algorithm produces $fp = fn$, which makes recall equal precision. Moreover, the recall rate that is computed from a specific dataset is certain and robust enough to compare the performance of the algorithm. We produce recall rate that computed from a standard dataset and our public dataset for the future comparison.

6.8 Discussion

In PETS09, using the spatial-only and texture-only returned a recall rate of 62% in both cases. Many people wear similar clothes which led to indistinguishable texture and low recall rate of texture-only data association when compared to DEC11 dataset.

In the disjoint network DEC11, the subject's position was scattered and the texture was more distinguishable compared to PETS09. This resulted in higher recall rate from the texture-only cost function compared to the spatial-only experiment.

In PETS09, the combined cost function gave a better recall rate compared to the use of individual cost functions. In DEC11, the best configuration is $\xi = 0$, which suggests that the trajectory spatial information does not contribute to the best result. The combined cost function is in a general form to switch between the spatial and the texture information.

The best combining factor ξ was varied in different situations. In PETS09 people have similar texture appearance compared to DEC11. In PETS09, spatial and texture are both important for data association. In contrast DEC11 has a significant spatial gap

between FOVs and this gap makes subject location uncertain. Therefore, the combining factor ξ has to be varied to choose whether to emphasise texture or spatial information. The variance σ , which comes from the stochastic model does not change much from PETS09 to DEC11 . Figure 6.10 and 6.11 shows similar patterns and we believe this pattern will be consistent in all dataset.

It require further study to confirm the consistency of the recall profile. The parameters (ξ, ν) must be adjusted from case to case for the optimum accuracy. It may be necessary to use an on-line learning algorithm to adjust the parameters.

In conclusion, we show the new appearance descriptor which is consistence across all cameras in the network. The consistency of descriptor allows matching between previous trajectory and a new subject. Furthermore, it can link the broken trajectories of many subjects to recreate the complete paths.

Chapter 7

Conclusion

From our review of tracking methods, a tracking method consists of the appearance descriptor and the state estimator. We can use a combination of various appearance descriptors as simple as edges or a 3D model. In order to integrate observations from multiple-cameras, we use a 3D ellipsoid model. Our ellipsoid projection transforms an ellipsoid surface in a 3D world to a quadratic equation in an image plane. The quadratic function makes silhouette likelihood computation faster than a conventional vertex base. In addition, memory utilisation is reduced dramatically. We choose the SIR particle filter for the state estimator because it keeps details of probability densities compared to the maximum-likelihood and maximum-a-posteriori methods. The particle filter is based on the MCMC method. A difference of the particle filter from the MCMC is that the particle filter generates multiple-sampling at the same time unlike the sample-chain in the MCMC. Multiple-sampling makes the particle approach suitable in parallel processing. We include position, velocity, visibility and persistence into the state vector. The position and velocity are used for predicting the future position. The visibility and persistence are designed for estimating existence of the subject.

Contribution 1: Likelihood from ellipsoid projection We show that the transformation from a 3D ellipsoid to a 2D quadratic function can accelerate likelihood computation compared to the conventional vertex base as described in Section 3.2. The silhouette likelihood is the similarity between a subject silhouette and the quadratic

equation to form an ellipse. The quadratic function is also used to determine interaction between multiple-targets. The intersection between subjects in image plane can be detected by deterministic ellipse functions. This allows us to estimate the sharing of a foreground pixel between multiple-subjects as described in Section 3.3.2. The membership probability of a pixel belonging to an owner subject is modeled to suppress the distraction problem. Distraction suppression allows the tracking system to learn the texture appearance of the subject for long enough to generate a texture signature. Once the texture signature is made the texture likelihood is applied. The texture likelihood increases the accuracy of tracking significantly and can be used in recognition to link the sub-trajectories into a complete trajectory.

Contribution 2: Analysis of the tracking framework In Chapter 4, we implemented the tracking system in C++ and tested on a single core of the CPU with 3.1GHz. We measure computation time for individual sub-functions. `Detection()` and `Likelihood()` are the two slowest sub-functions. The computation time of `Detection()` increases linearly with the size of the image and the number of grids. Whereas the computation time of `Likelihood()` is proportional to the number of particles, the number of active pixels and the number of targets. Increasing the number of particles reduces the rate of breaking of a trajectory but there is no significant improvement in false-alarm and miss-detection rates. We also implemented the detection module to fulfill an automatic detection and tracking.

We also tested the tracking framework with standard dataset and compared the MOTA as showed in Table 4.4. Our approach produces higher accuracy compared the other methods tested by the PETS09 dataset. The improvement comes from invariant 3D shape and texture signature which makes each subject distinguishable.

Contribution 3: GPU acceleration The sequential tracking program is transferred to CUDA code, which is executed on a GPU with 128 cores at clock speed of 1.6 GHz. We transferred all sub-functions to CUDA and measured the computation times. We applied 4 methods (data parallelism, reduction, skip ahead and local memory) to optimise the computation time. The GPU implementation is not fully optimised according to the

CUDA profiler. Serialisation and cache-miss are the major problems. However, we achieved 3.5 times speedup from the lower frequency GPU compared to the 3.1GHz CPU. The best configuration for real-time tracking using the particle filter is to process `Detection()` and `Likelihood()` on the GPU and the remaining sub-functions on the host. This pipe-lining on the host and device could improve the speedup ratio about 10%.

Contribution 4: Spatial-Texture data association In Chapter 6, we studied the data association problem in joint and disjoint camera networks. The data association links unknown subjects with pre-existing trajectories based on cost function. In order to compare effectiveness of using between texture signature and spatial distance, we conducted three experiments; spatial-only, texture-only and spatial-texture combination. The spatial cost function is defined by a Gaussian probability density, where the variance is determined by the stochastic model of motion, and the texture cost function is determined by the similarity between the colour-histogram of each horizontal section of the ellipsoid. The cost functions are filtered by the *Ramp* function and the results is used for constructing the cost matrix. The data association assignments between unknown subjects and pre-existing trajectories finally are computed by Hungarian method. The results show that the combination of spatial and texture improves the recognition rate. The combined cost function gave a higher or equal recall rate compared to the cost functions determined from spatial and texture separately. However, the combining factor vary from case to case. The combining factor should be adjusted adaptively to the situation.

7.1 Future work

Pan-tilt-zoom cameras This tracking framework was designed for a static camera network, however, the framework can be extended for fixed position pan-tilt-zoom cameras. The pan-tilt-zoom can be added to the camera projection model this extension makes the system more robust and versatile.

Appearance descriptors The primary input of our frame work is colour images, which is sensitive to lighting condition. The illumination tolerant descriptor, such as a gradient image, should used as the input instead of the colour images.

Optimising the GPU implementation At current state we made 3.5 times and the main problems are serialisation and cache-miss. The algorithm and implementation of likelihood function may need to be re-designed.

Adaptive combining factor for the spatial-texture cost function As the discussion in the Chapter 6, the best combining factor β depends on particular scenario. Therefore, ξ should be determined at real-time. We hope the adaptive combining factor should maintain recall rate around 80%.

Appendix A

LM Optimization

A.1 LM Optimization

Optimization is an interesting topic for many fields of study. In Chapter 6 camera calibration has been made by LM optimization. A well known non-linear (second order) algorithm Levenberg-Marquardt(LM) has been developed based on steepest descent and Gauss-Newton methods(see details in [188]).

A function $F = F(\mathbf{x})$ is a general cost function which is the squared-error. In camera calibration F is computed from the sum of error square in a back-projection image. The function F is written as a function of error (\mathbf{f}) as shown by Equation (A.1).

$$F(x) = \frac{1}{2} \sum |f_i(x_j)|^2 = \frac{1}{2} \mathbf{f}^T \mathbf{f} \quad (\text{A.1})$$

Note the Jacobian matrix $\mathbf{J} = \frac{dF}{d\mathbf{x}} = \frac{dF}{d\mathbf{f}} \frac{d\mathbf{f}}{d\mathbf{x}} = \mathbf{J}^T \mathbf{f}$. To estimate the local minimum point, the function F is expanded by Taylor's series.

$$\begin{aligned} F(x_o + h) &= F(x_o) + F'(x_o)h + \frac{1}{2} h^T F''(x_o)h + O(|h|^3) \\ F'(x_o) &= \mathbf{J}_o^T \mathbf{f} \\ F''(x_o) &= H_o \end{aligned}$$

The matrix \mathbf{J}_o is a Jacobian matrix of \mathbf{f} at $x = x_o$ and the inner product $\mathbf{J}_o^T \mathbf{J}_o = H_o$ is known as a Hessian matrix. \mathbf{h} is an update step whose value is calculated by the following techniques.

STEEPEST DESCENT METHOD The update step \mathbf{h} is a descent direction for F at x , which makes $\mathbf{h}^T F'(x) < 0$. The update direction depends on the gradient of function F , which can be calculated by the Jacobian matrix of F at particular location x . Table A.1 shows a summary of the steepest descent algorithm.

```

for( $t = 1 : t_{max}$ )
   $\mathbf{h}_{sd} = -\mathbf{J}^T \mathbf{f}$ ;
   $\mathbf{x}_{t+1} = \mathbf{x}_t + \alpha \mathbf{h}_{sd}$ ;
end

```

TABLE A.1: Steepest descent algorithm

GAUSS-NEWTON METHOD At a stationary point, the first derivative of a scalar function $y(x)$ is zero $y'(x) = 0$, so the optimum point can be solved by Newton's method as shown by Equation (A.2). This is known as the Gauss-Newton method.

$$\begin{aligned}
 x_{t+1} &= x_t - \frac{y'(x_t)}{y''(x_t)} \\
 h_{gn} &= -\frac{y'(x_i)}{y''(x_i)} \\
 h_{gn} y''(x_i) &= -y'(x_i)
 \end{aligned} \tag{A.2}$$

Concerning the multi-dimension function \mathbf{F} , the update step \mathbf{h} can be computed by the Gauss-Newton algorithm as in Table A.2.

```

for( $t = 1 : t_{max}$ )
  Solve :  $(\mathbf{J}^T \mathbf{J}) h_{gn} = -\mathbf{J}^T \mathbf{f}$ ;
   $\mathbf{x}_{t+1} = \mathbf{x}_t + \alpha \mathbf{h}_{sd}$ ;
end

```

TABLE A.2: Gauss-Newton algorithm

LM METHOD When \mathbf{x} is far from a local minimum point, using steepest descent is better than the Gauss-Newton method because the second-order of Taylor's series produces a large error, while steepest decent always guarantees that its update direction is correct (but the magnitude of the update vector is still unknown). In contrast when \mathbf{x} is close to the optimum point, the surface of the cost function is very smooth and fits to the second-order of the Taylor's series, so Gauss-Newton produce *superlinear convergence* [188]. At this condition Gauss-Newton converges to a local minimum point faster than the steepest descent method. The LM method combines both benefits by adding a *damping term* in the Gauss-Newton method. It enables the algorithm to use both steepest descent and Gauss-Newton characteristics. Levenberg and Marquardt's numerical formula is shown in Equation (A.3).

$$(\mathbf{J}^T \mathbf{J} + \mu \mathbf{I}) \mathbf{h}_{new} = -\mathbf{J}^T \mathbf{f} \quad (\text{A.3})$$

The damped term $\mu \mathbf{I}$ (identity matrix \mathbf{I}) is a switch to alternate between the two methods. When the damping parameter μ is small the first term (Hessian) dominates the left hand side. So, Equation (A.3) is approximately equal to the Gauss-Newton method. For large, μ the equation is approximately the same as the steepest descent method.

```

 $\mu = 0.01; \mathbf{h}_{lm} = \mathbf{h}_o;$ 
for( $t = 1 : t_{max}$ )
  Solve :  $(\mathbf{J}^T \mathbf{J} + \mu \mathbf{I}) \mathbf{h}_{new} = -\mathbf{J}^T \mathbf{f};$ 
  if  $F(\mathbf{x}_t + \mathbf{h}_{new}) < F(\mathbf{x}_t + \mathbf{h}_{lm})$ 
     $\mathbf{h}_{lm} = \mathbf{h}_{new};$ 
     $\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{h}_{lm};$ 
     $\mu = \frac{\mu}{10};$ 
  else
     $\mu = 10\mu;$ 
  end
end

```

TABLE A.3: LM algorithm

In order to adapt the damping parameter μ , the values of the cost function F at the current step, $F(\mathbf{x}_t)$, and the next step, $F(\mathbf{x}_{t+1})$, are compared. If $F(x_{t+1}) \geq F(x_t)$, the damping value μ is increased. When $F(x_{t+1}) < F(x_t)$ this means that \mathbf{x} is moving closer to a local minimum, so μ is decreased to enable the characteristic of the Gauss-Newton method. The LM algorithm has been summarized in Table A.3, where $\mathbf{J} = \mathbf{J}(x, h_{lm})$ and $\mathbf{f} = \mathbf{f}(h_{lm})$. The LM method is stable enough to refine camera parameters [189]. A better version of LM method is summarised in [188].

The LM method should start with good guess or it is possibly to be stuck at a local minimum. Therefore the LM method should be used when some parameters are known (such as intrinsic parameters) and extrinsic parameter can be presumed by the user.

To implement this method we computed the Jacobian by using a symbolic toolbox in MATLAB. The Jacobian expression in symbolic terms is too long to show in this section. The Matlab code for generating the Jacobian of the camera projection matrix is shown in Listing A.1. The code will generate an in-line function and is named as `getJ()`.

```

1 %genJ.m--to compute Jacobian matrix of camera projection
2 %The process starts with declaring parameters as symbolic variables
3 %Intrinsic parameters are centre coordinate(uo, vo), scaled focals
4 %(\alpha \beta) and skewness (s)
5 syms uo vo \alpha \beta s real
6 %translation t and a Rodrigues rotation vector w
7 syms tx ty tz wx wy wz real
8 %A pair of key points in realworld coordinate (M) and image coordinate (m)
9 syms Mx My mx my real
10 syms scale real
11 %define the function
12 K=[ \alpha      s      uo;
13      0          \beta   vo;
14      0          0      1];
15 theta=sqrt(wx^2+wy^2+wz^2);
16 omega=[ 0  -wz   wy;
17         wz   0  -wx;
18        -wy  wx   0];
19 R=eye(3)+sin(theta)*omega/theta+(1-cos(theta))*omega^2/theta^2;
20 %define translation matrix
21 t=[tx;ty;tz];
22 %define projection matrix
23 m=K*[R(:,1) R(:,2) t]* [Mx;My;1];
24 %define coordinates on image plane
25 mx_est=m(1,1)/m(3,1);
26 my_est=m(2,1)/m(3,1);
27 %define error
28 f=[mx-mx_est;my-my_est];
29 %solve Jacobian
30 Js=jacobian(f, [\alpha,\beta,s,tx,ty,tz,uo,vo,wx,wy,wz]);
31 %generate an inline function
32 getJ=inline(Js);

```

LISTING A.1: A code to generate Jacobian matrix

Appendix B

Appendix DEC11 dataset

In this section, we will describe the procedure for creating the DEC11 dataset what created in December 2011. We set cameras around the small park near the EM building in Heriot-Watt University. It was cold and the illumination conditions changed very rapidly as normal as is with Scotland weather. The winter lighting made a dataset even more challenging for the tracking framework; where the sun is near the horizon this caused a very long shadow from each subject.

We used 6 camcorders (Cannon Legria HF S20) to produce this dataset. The layout of cameras relative to the ground floor is illustrated in Figure [B.1](#). In order to create a dataset for 3D tracking we need synchronised image sequences and camera calibration parameters from all cameras. We manually synchronised the image sequences and calibrated the intrinsic parameters. The procedure below is the method that we used to get the extrinsic parameters.

B.1 Generating Landmarks

The procedure starts by creating many landmarks on the ground plane. A camera must see at least 3 landmarks in order to determine the extrinsic camera parameters. Initially, we prepared a coloured sticky tape to create markers on the ground plane. However, the ground was wet due to snow. As the ground was difficult for mounting markers,

we decided to use the locations of existing static objects such as pavements or solid structures. All lengths between two markers were measured to construct a network graph of nodes and edges. In order to determine the exact coordinates of a landmark node, we needed at least 2 edges. A graph of the collection of nodes and edges was stored in xml file and used for refining the coordinates on the ground plane.

B.2 Refining coordinates

To determine the x-y coordinate of a node on the ground plane, we defined energy in the network between two nodes as the square of difference between the measurement and current state. This is similar to a network of springs, where the length at the equilibrium of the spring is substituted by the measurement lengths between nodes. If we leave the spring network to adjust itself during iteration, the system will minimise the total energy and the final state will have minimum energy (square of error). Suppose \mathbf{D}_{BA} is a vector pointing from a node A to a node B . The measurement of a scalar value of length is denoted by L_{BA} , which was measured from the scene. Then the force acting on the node B by the edge BA is

$$\mathbf{F}_{BA} = \mathbf{D}_{BA} - L_{BA} \cdot \frac{\mathbf{D}_{BA}}{|\mathbf{D}_{BA}|} \quad (\text{B.1})$$

$$\mathbf{F}_{BA} = \mathbf{D}_{BA} \left(1 - \frac{L_{BA}}{|\mathbf{D}_{BA}|} \right) \quad (\text{B.2})$$

After embedding the network and forces of each node into a program (RefPoints/-main.m) we then let the process minimise the energy for 10,000 iteration. The resulting graph had a sum of square-error approximately $5.4 \times 10^{-4} [mm]^2$ in the final state. The iteration process and final landmark coordinates are illustrated in Figure B.3 and Figure B.2.

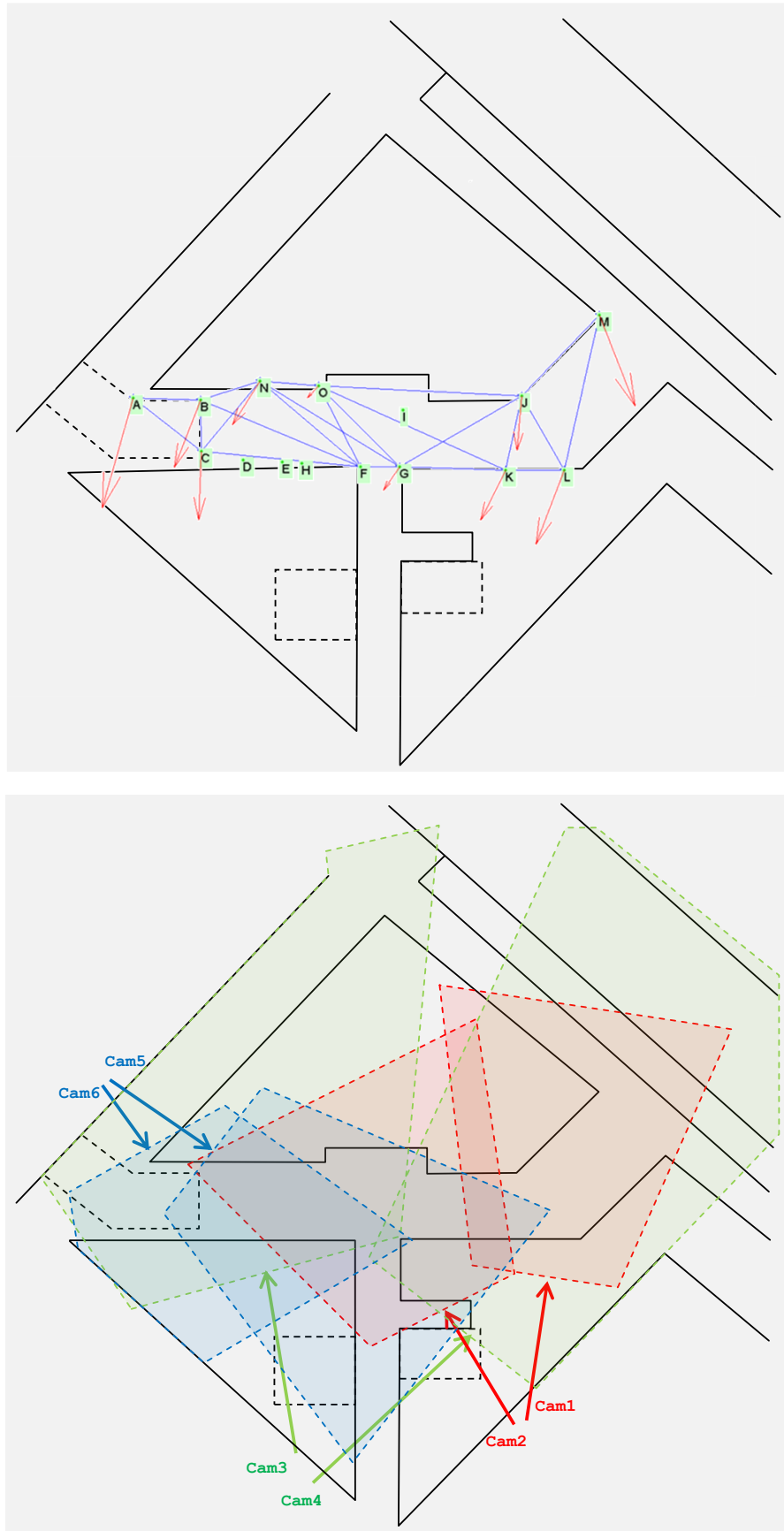


FIGURE B.1: (top) Landmarks and (bottom) Camera views.

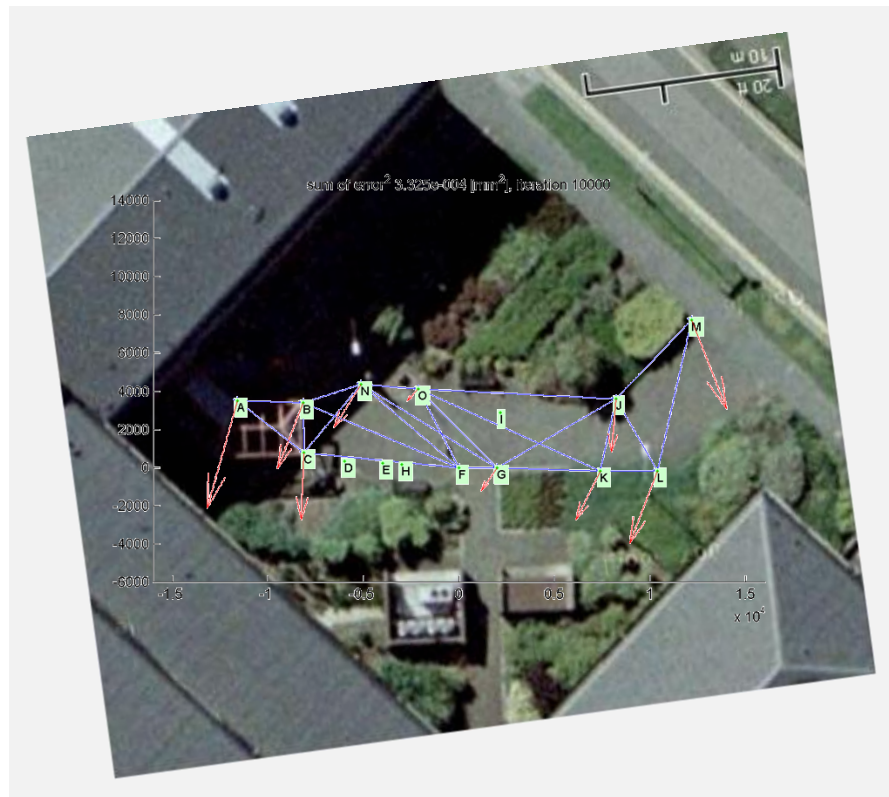


FIGURE B.2: Overlay of the landmark coordinates on a Google map.

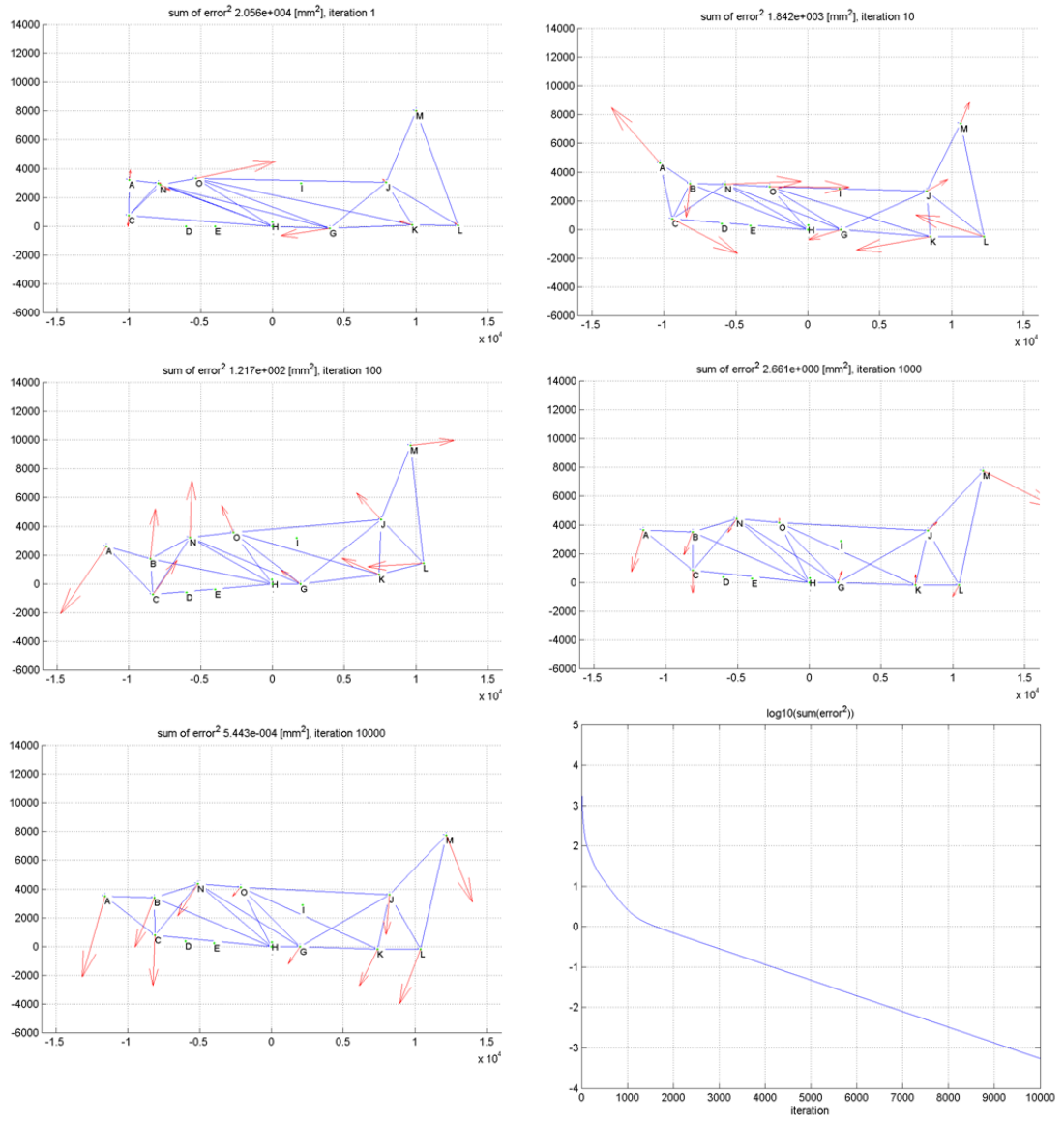


FIGURE B.3: Iteration process to refine the landmark coordinates.

B.3 Extrinsic Calibration

Normally we can use the Matlab calibration toolbox [136] to estimate both the intrinsic and extrinsic parameters by using a chessboard. A chessboard provides landmarks but we could not use a chessboard in the outdoor scenarios because of the huge area. To work with a chessboard of size of $10 \times 10m^2$ would be impractical. For outdoor calibration, the following method is more suitable.

The process starts with calibrating the intrinsic parameters by using the standard toolbox [136]. Then we stored the intrinsic matrix for subsequent processes. The homography matrix was computed as described in Section 2.6.1. We could find the extrinsic parameters from the homography directly but we have a small number of landmarks, which leads to uncertainty of estimation. The solution of extrinsic parameters computed by homography only gave an inaccurate result as in Figure B.8. So, we used Levenburg-Marquardt (LM) optimisation to reduce the residual error. The details of LM optimisation are given in Appendix A.

The resulting back-projection and camera orientation are shown in Figures B.4 to B.9. In the back-projection images (top and middle) the red-green-blue axes are the x, y and z directions in the 3D world. The red circles in back-projection images are landmark locations, whilst the blue markers show projected locations. The camera orientations are illustrated in the bottom images, where the blue dot is the origin of the 3D coordinate system and the red-green-blue axes are the directions of the camera coordinates.

We tried the procedure of extrinsic calibration in practice and noticed that the standard chessboard method was impractical in the outdoor scenarios. Our solution was to exploit existing landmarks in the field. The process allowed us to determine the orientation and position of cameras after the incident taking place. Investigations after an incident can be made even if the cameras have never been calibrated before.

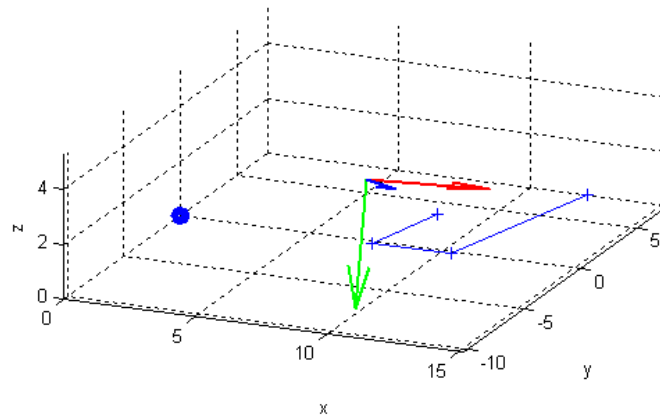


FIGURE B.4: Projection of Camera1, (top) solution by using homography (middle) by LM optimisation and (bottom) position of camera.



FIGURE B.5: Projection of Camera2, (top) solution by using homography (middle) by LM optimisation and (bottom) position of camera.

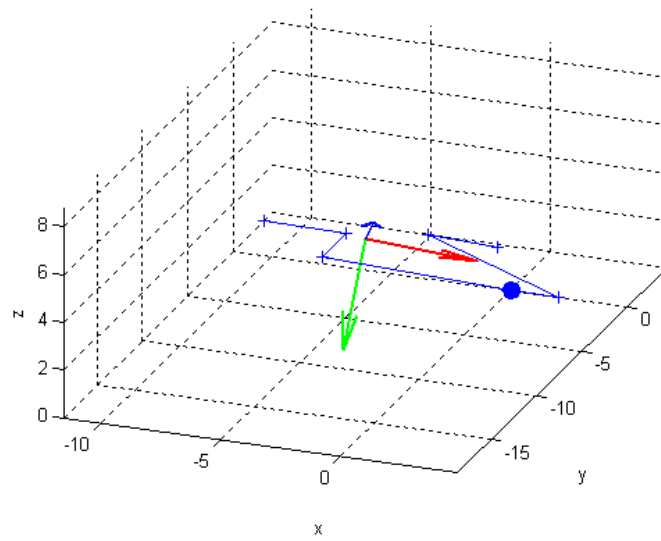
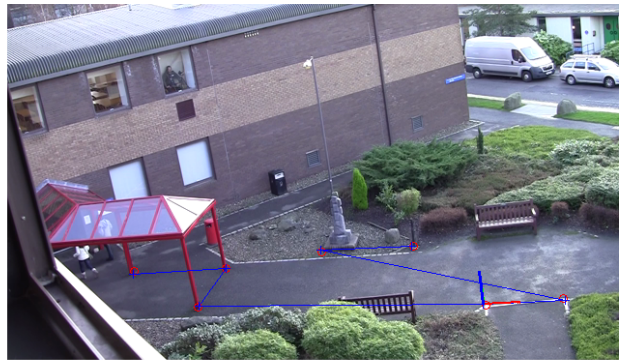
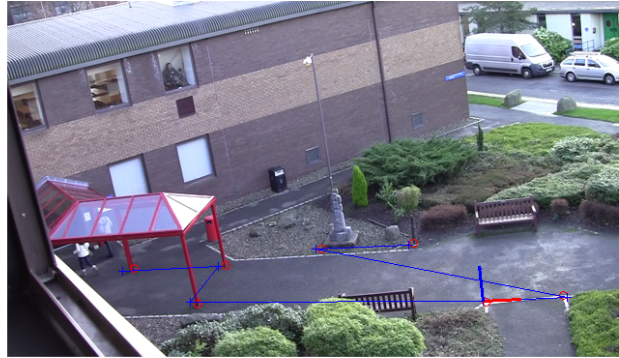


FIGURE B.6: Projection of Camera3, (top) solution by using homography (middle) by LM optimisation and (bottom) position of camera.

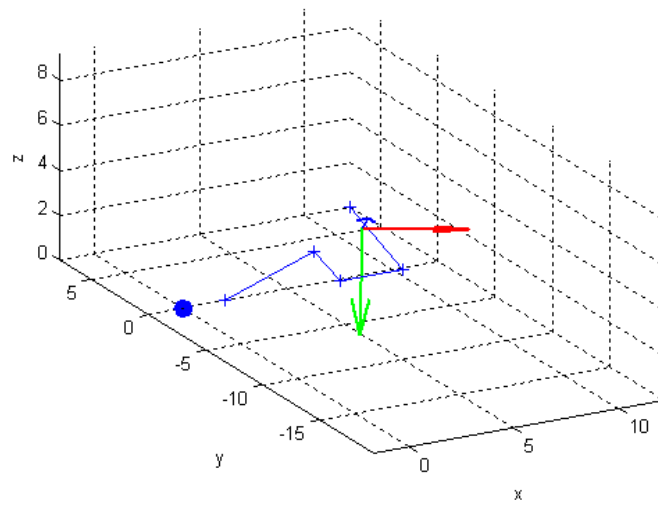


FIGURE B.7: Projection of Camera4, (top) solution by using homography (middle) by LM optimisation and (bottom) position of camera.

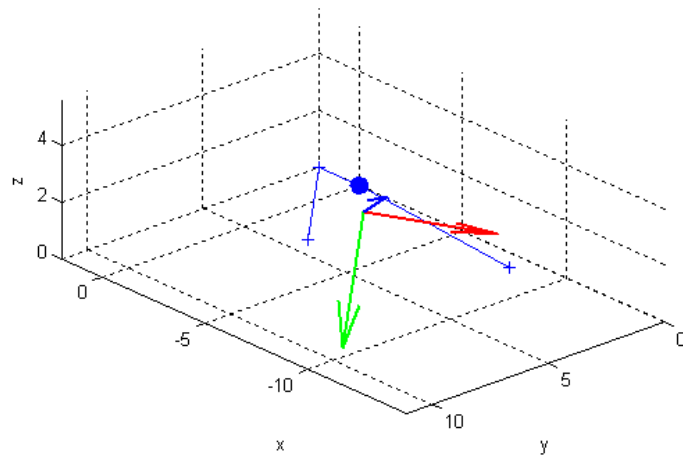


FIGURE B.8: Projection of Camera5, (top) solution by using homography (middle) by LM optimisation and (bottom) position of camera.

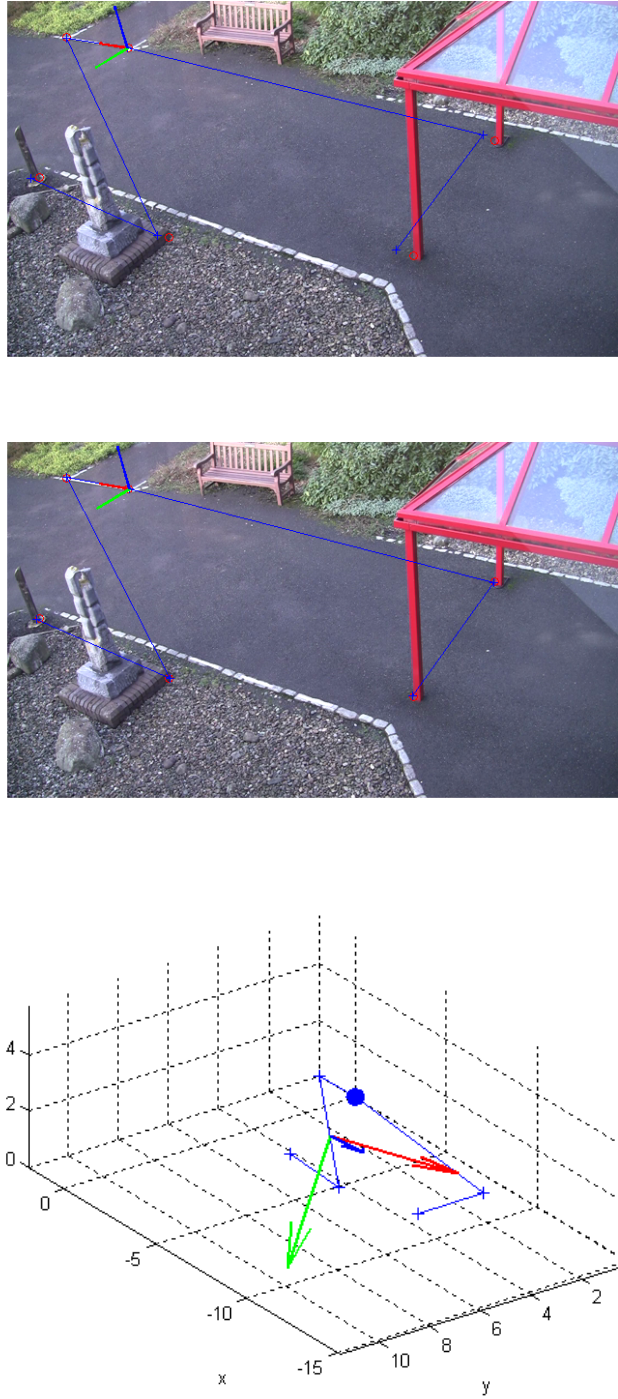


FIGURE B.9: Projection of Camera6, (top) solution by using homography (middle) by LM optimisation and (bottom) position of camera.

B.4 Calibration parameters DEC11

A translation vector \mathbf{t} is expressed in a unit metre.

Camera1

$$\mathbf{K} = \begin{bmatrix} 754.8575 & 0 & 283.5323 \\ 0 & 769.8163 & 153.9865 \\ 0 & 0 & 1.0000 \end{bmatrix} \quad (\text{B.3})$$

$$\mathbf{R}|\mathbf{t} = \begin{bmatrix} 0.9877 & 0.1558 & 0.0119 & -9.3410 \\ 0.0686 & -0.3643 & -0.9288 & 0.9903 \\ -0.1404 & 0.9182 & -0.3705 & 11.3563 \end{bmatrix} \quad (\text{B.4})$$

Camera2

$$\mathbf{K} = \begin{bmatrix} 750.7645 & 0 & 286.1683 \\ 0 & 765.3397 & 159.2458 \\ 0 & 0 & 1.0000 \end{bmatrix} \quad (\text{B.5})$$

$$\mathbf{R}|\mathbf{t} = \begin{bmatrix} 0.7626 & 0.6467 & -0.0120 & -0.1302 \\ 0.2391 & -0.2991 & -0.9238 & -0.1149 \\ -0.6010 & 0.7016 & -0.3828 & 14.6948 \end{bmatrix} \quad (\text{B.6})$$

Camera3

$$\mathbf{K} = \begin{bmatrix} 751.5750 & 0 & 289.5567 \\ 0 & 766.5949 & 153.1718 \\ 0 & 0 & 1.0000 \end{bmatrix} \quad (\text{B.7})$$

$$\mathbf{R|t} = \left[\begin{array}{ccc|c} 0.9579 & 0.2317 & -0.1697 & 5.0394 \\ -0.1054 & -0.2661 & -0.9582 & 3.8177 \\ -0.2672 & 0.9357 & -0.2305 & 18.5635 \end{array} \right] \quad (\text{B.8})$$

Camera4

$$\mathbf{K} = \begin{bmatrix} 752.5889 & 0 & 313.6071 \\ 0 & 768.3470 & 145.2730 \\ 0 & 0 & 1.0000 \end{bmatrix} \quad (\text{B.9})$$

$$\mathbf{R|t} = \left[\begin{array}{ccc|c} 0.8450 & -0.5348 & -0.0020 & -8.4613 \\ -0.1397 & -0.2171 & -0.9661 & 4.9666 \\ 0.5162 & 0.8166 & -0.2581 & 16.9646 \end{array} \right] \quad (\text{B.10})$$

Camera5

$$\mathbf{K} = \begin{bmatrix} 752.5889 & 0 & 313.6071 \\ 0 & 768.3470 & 145.2730 \\ 0 & 0 & 1.0000 \end{bmatrix} \quad (\text{B.11})$$

$$\mathbf{R|t} = \left[\begin{array}{ccc|c} -0.8005 & -0.5920 & -0.0933 & -2.6315 \\ -0.1034 & 0.2897 & -0.9515 & 1.3012 \\ 0.5904 & -0.7521 & -0.2931 & 15.6003 \end{array} \right] \quad (\text{B.12})$$

Camera6

$$\mathbf{K} = \begin{bmatrix} 760.8852 & 0 & 319.4545 \\ 0 & 775.4238 & 157.1560 \\ 0 & 0 & 1.0000 \end{bmatrix} \quad (\text{B.13})$$

$$\mathbf{R|t} = \left[\begin{array}{ccc|c} -0.8350 & -0.5342 & -0.1317 & -3.8247 \\ -0.1935 & 0.5092 & -0.8386 & -2.1201 \\ 0.5151 & -0.6748 & -0.5285 & 15.1663 \end{array} \right] \quad (\text{B.14})$$

Appendix C

EM330 dataset

In this section we describe the procedure to created the EM330 dataset, which is a synchronised image sequence captured by 3 cameras. The camera layout was set as a joint camera network as shown in Figure C.1. All digital video signals were sent to a host computer via the IEEE-1394b protocol.

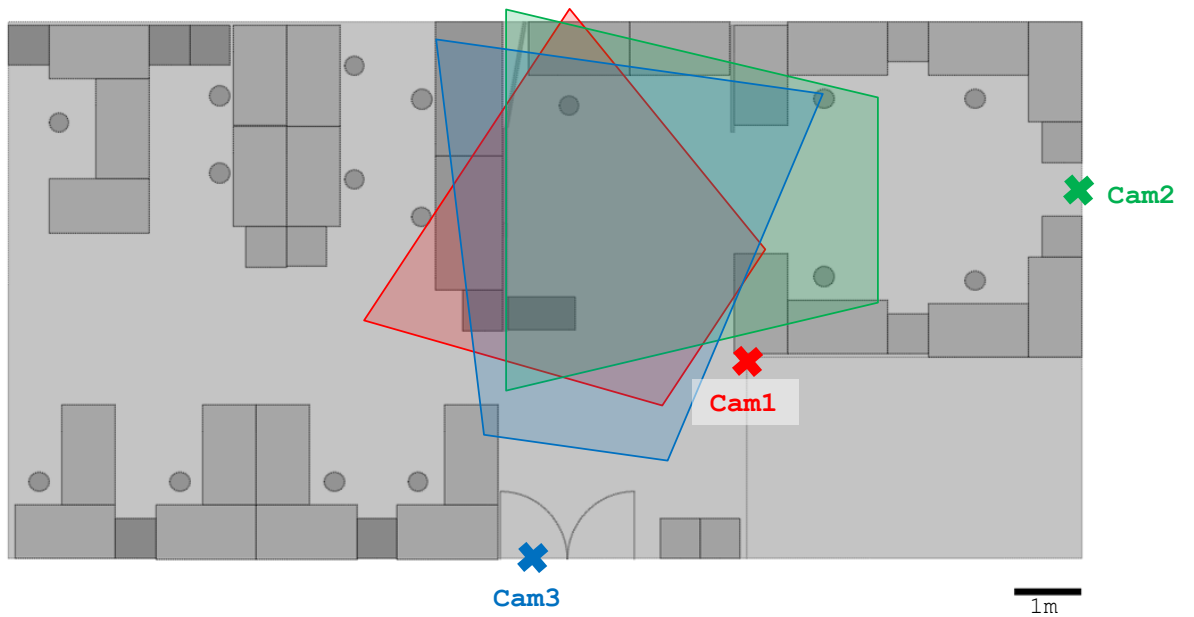


FIGURE C.1: Camera layout in EM330

C.1 Hardware and software requirements

The cameras were PointGrey FL2-14S3C mounted with Fujinon 6mm f1.2 lenses. The cameras were connected to host by an IEEE1394b adapter card, which can handle up to 160Mbps. However, the synchronizing forced all cameras to push data to the host at the same period. So we were able to connect at lowest frame rate (7.5fps) with RGB 640×480 resolution. The original setting had 3 cameras and later we extended to 4 cameras for a demonstration of real-time tracking.

In order to load the video data, we used the PointGrey API library. The library allowed us to configure the internal camera parameters, such as white-balance and gamma value, and also provided a data loading interface. The loading interface stored an image from each camera at a particular frame in the internal memory of the host computer and the data address (a pointer) was used in order to pass the image to host functions or function on the GPU.

In this dataset we used the Matlab calibration toolbox [136] and a big chessboard. The chessboard had 12cm black and white squares. The squares had to be large because the cameras were far from the chessboard and the squares appeared about 10 pixels in the image plane.

C.2 Calibration parameters EM330

Camera1

$$\mathbf{K} = \begin{bmatrix} 673.1468 & 0 & 332.0133 \\ 0 & 670.7951 & 222.0849 \\ 0 & 0 & 1.0000 \end{bmatrix} \quad (\text{C.1})$$

$$\mathbf{R}|\mathbf{t} = \begin{bmatrix} 0.9114 & -0.4107 & -0.0257 & 1.0152 \\ -0.1843 & -0.3516 & -0.9178 & 1.3909 \\ 0.3679 & 0.8413 & -0.3961 & 3.9112 \end{bmatrix} \quad (\text{C.2})$$

Camera2

$$\mathbf{K} = \begin{bmatrix} 655.0845 & 0 & 328.4335 \\ 0 & 650.5366 & 237.2510 \\ 0 & 0 & 1.0000 \end{bmatrix} \quad (\text{C.3})$$

$$\mathbf{R|t} = \left[\begin{array}{ccc|c} 0.9995 & 0.0122 & -0.0277 & -0.0779 \\ -0.0203 & -0.4094 & -0.9121 & -0.2226 \\ -0.0224 & 0.9123 & -0.4089 & 7.1536 \end{array} \right] \quad (\text{C.4})$$

Camera3

$$\mathbf{K} = \begin{bmatrix} 657.9701 & 0 & 342.0420 \\ 0 & 658.9150 & 221.4482 \\ 0 & 0 & 1.0000 \end{bmatrix} \quad (\text{C.5})$$

$$\mathbf{R|t} = \left[\begin{array}{ccc|c} -0.3760 & -0.9266 & 0.0014 & -0.7516 \\ -0.5259 & 0.2122 & -0.8237 & -0.3734 \\ 0.7629 & -0.3104 & -0.5671 & 5.5340 \end{array} \right] \quad (\text{C.6})$$

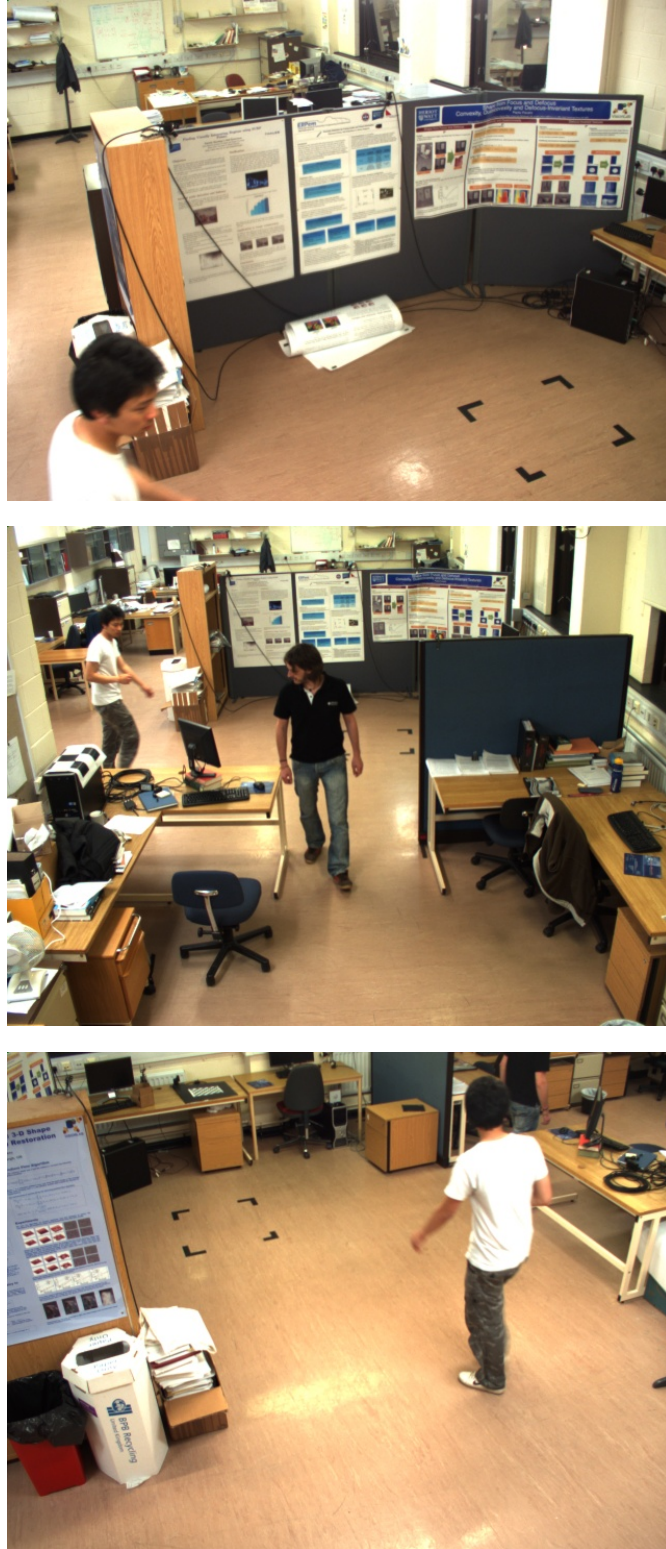


FIGURE C.2: Top to bottom are sample images from camera1, camera2 and camera3

Appendix D

Distance and similarity

TABLE D.1: List of distance and similarity formulas

Name	Distance	Similarity
Euclidean	$(\sum_i (X_i - Y_i)^2)^{\frac{1}{2}} = X - Y $	
Square Euclidean	$\sum_i (X_i - Y_i)^2$	
Manhattan (Taxicap)[11]	$\sum_i X_i - Y_i $	
Chebyshev (Chessboard)[12]	$\max_i (X_i - Y_i)$	
Cosine	$1 - \frac{\sum_i X_i Y_i}{ X \cdot Y }$	$\frac{\sum_i X_i Y_i}{ X \cdot Y }$
Correlation[13]		$\sum_i (X_i - \bar{X})(Y_i - \bar{Y})$
Bhattacharyya[14]	$-\log(\sum_i \sqrt{x_i y_i})$	$\sum_i \sqrt{x_i y_i}$
Hamming[15]	$\sum_i \text{notequal}(A_i, B_i)$	
Jaccard[16]	$1 - \frac{ A \cap B }{ A \cup B }$	$\frac{ A \cap B }{ A \cup B }$
Hausdorff[17]	$\max(h(A, B), h(B, A)),$ $h(A, B) = \max_i(\min_j(A_i, B_j))$	

Bibliography

- [1] Jean Piaget. Piaget's theory. In *Handbook of Child Psychology*, volume 1. New York: Wiley, 4 edition, 1983.
- [2] George K. Smelser. The retina. *Archives of Ophthalmology*, 27(2):430–430, 1942. doi: 10.1001/archopht.1942.00880020216024. URL [+http://dx.doi.org/10.1001/archopht.1942.00880020216024](http://dx.doi.org/10.1001/archopht.1942.00880020216024).
- [3] Eichi Yamada. Some structural features of the fovea centralis in the human retina. *Archives of Ophthalmology*, 82(2):151–159, 1969. doi: 10.1001/archopht.1969.00990020153002. URL [+http://dx.doi.org/10.1001/archopht.1969.00990020153002](http://dx.doi.org/10.1001/archopht.1969.00990020153002).
- [4] John Findlay and Robin Walker. Human saccadic eye movements. *Scholarpedia*, 6(11):5095, 2011.
- [5] Richard J. Krauzlis. Recasting the smooth pursuit eye movement system. *Journal of Neurophysiology*, 91(2):591–603, 2004. doi: 10.1152/jn.00801.2003. URL <http://jn.physiology.org/content/91/2/591.abstract>.
- [6] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *INTERNATIONAL JOURNAL OF COMPUTER VISION*, 1(4):321–331, 1988. URL <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.124.5318>.
- [7] Zezhi Chen and Andrew M. Wallace. Active segmentation and adaptive tracking using level sets. In *BMVC*, 2007.

- [8] Rolf H. Baxter, Neil M. Robertson, and David M. Lane. Real-time event recognition from video via a bag-of-activities. In *Proceedings of the UAI Bayesian Modelling Applications Workshop*, 2011.
- [9] Roy Longbottom. Cuda gpu parallel computing benchmarks, January 2011. URL <http://www.roylongbottom.org.uk/cuda1.htm>.
- [10] PCPlus. Intel core i7 benchmarks, 2008. URL <http://mos.techradar.com/techradar-corei7-benchmarks.pdf>.
- [11] Margherita Barile. Taxicab metric. From MathWorld—A Wolfram Web Resource, created by Eric W. Weisstein. URL <http://mathworld.wolfram.com/TaxicabMetric.html>.
- [12] James M. Abello and Panos M. Pardalos. *Handbook of Massive Data Sets*. Springer, 2002.
- [13] Eric Weisstein. Correlation coefficient, 2012. URL <http://mathworld.wolfram.com/CorrelationCoefficient.html>.
- [14] A. Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society*, 35:99–109, 1943.
- [15] Richard W. Hamming. Error detecting and error correcting codes, 1950.
- [16] Maureen Hillenmeyer. Jaccard coefficient, 2006. URL <http://www.stanford.edu/~maureenh/quals/html/ml/node68.html>.
- [17] D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. Comparing images using the hausdorff distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(9):850–863, sep 1993. ISSN 0162-8828. doi: 10.1109/34.232073.
- [18] Greg Welch and Gary Bishop. An introduction to the kalman filter. Technical report, Chapel Hill, NC, USA, 1995.

- [19] C. Kreucher, K. Kastella, and A. O. Hero. Multitarget tracking using the joint multitarget probability density. 41(4):1396–1414, Oct. 2005. doi: 10.1109/TAES.2005.1561892.
- [20] Jianqing Fan and Jinchi Lv. A selective overview of variable selection in high dimensional feature space (invited review article). October 2009. URL <http://arxiv.org/abs/0910.1122>.
- [21] D. Avitzour. A maximum likelihood approach to data association. *Aerospace and Electronic Systems, IEEE Transactions on*, 28(2):560–566, apr 1992. ISSN 0018-9251. doi: 10.1109/7.144581.
- [22] R L Streit and T E Luginbuhl. Probabilistic multi-hypothesis tracking. Technical report, Naval Undersea Warfare Center Division Newport, Rhode Island, 1995.
- [23] Michael Isard and Andrew Blake. Condensation - conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29:5–28, 1998.
- [24] J.A. Brown and D.W. Capson. A framework for 3d model-based visual tracking using a gpu-accelerated particle filter. *Visualization and Computer Graphics, IEEE Transactions on*, 18(1):68–80, jan. 2012. ISSN 1077-2626. doi: 10.1109/TVCG.2011.34.
- [25] Jongwoo Lim, David Ross, Ruei sung Lin, and Ming hsuan Yang. Incremental learning for visual tracking. In *In Advances in Neural Information Processing Systems*, pages 793–800. MIT Press, 2004.
- [26] Z. Kalal, K. Mikolajczyk, and J. Matas. Face-tld: Tracking-learning-detection applied to faces. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pages 3789–3792, sept. 2010. doi: 10.1109/ICIP.2010.5653525.
- [27] Anurag Mittal and Larry S. Davis. M2tracker: A multi-view approach to segmenting and tracking people in a cluttered scene using region-based stereo. In *International Journal of Computer Vision*, pages 189–203, 2002.
- [28] Hossam Osman, Li Pan, Steven D. Blostein, and Langis Gagnon. Classification of ships in airborne sar imagery using backpropagation neural networks, 1997.

- [29] Eric W. Weisstein. Maximum likelihood, . URL <http://mathworld.wolfram.com/MaximumLikelihood.html>.
- [30] R. E. Kalman. A new approach to linear filtering and prediction problems. 1960. URL <http://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf>.
- [31] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8(6):679–698, nov. 1986. ISSN 0162-8828. doi: 10.1109/TPAMI.1986.4767851.
- [32] G. Unal, H. Krim, and A. Yezzi. Active polygon for object tracking. In *3D Data Processing Visualization and Transmission, 2002. Proceedings. First International Symposium on*, pages 696 – 699, 2002. doi: 10.1109/TDPVT.2002.1024143.
- [33] Hyung-Bok Kim and Kwee-Bo Sim. A particular object tracking in an environment of multiple moving objects. In *Control Automation and Systems (ICCAS), 2010 International Conference on*, pages 1053 –1056, oct. 2010.
- [34] Yazhe Tang, Youfu Li, Tianxiang Bai, Xiaolong Zhou, and Zhongwei Li. Human tracking in thermal catadioptric omnidirectional vision. In *Information and Automation (ICIA), 2011 IEEE International Conference on*, pages 97 –102, june 2011. doi: 10.1109/ICINFA.2011.5948970.
- [35] P. Hough and B. Powell. A method for faster analysis of bubble chamber photographs. *Il Nuovo Cimento (1955-1965)*, 18:1184–1191, 1960. ISSN 1827-6121. URL <http://dx.doi.org/10.1007/BF02733175>. 10.1007/BF02733175.
- [36] Guorong Li, Wei Qu, and Qingming Huang. A multiple targets appearance tracker based on object interaction models. *Circuits and Systems for Video Technology, IEEE Transactions on*, 22(3):450 –464, march 2012. ISSN 1051-8215. doi: 10.1109/TCSVT.2011.2165591.
- [37] M. Vincze, M. Ayromlou, and M. Zillich. Fast tracking of ellipses using edge-projected integration of cues. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 4, pages 72 –75 vol.4, 2000. doi: 10.1109/ICPR.2000.902867.

- [38] T. Ardeshiri, F. Larsson, F. Gustafsson, T.B. Schon, and M. Felsberg. Bicycle tracking using ellipse extraction. In *Information Fusion (FUSION), 2011 Proceedings of the 14th International Conference on*, pages 1–8, july 2011.
- [39] Radim Halir and Jan Flusser. Numerically stable direct least squares fitting of ellipses, 1998.
- [40] D M Greig, B. T. Porteous, and A. H. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society. Series B (Methodological)*, 51:271–279, 1989.
- [41] P. Salembier, L. Torres, F. Meyer, and Chuang Gu. Region-based video coding using mathematical morphology. *Proceedings of the IEEE*, 83(6):843–857, jun 1995. ISSN 0018-9219. doi: 10.1109/5.387088.
- [42] Ralf Plaenkers and Pascal Fua. Model-based silhouette extraction for accurate people tracking. In *Proceedings of the 7th European Conference on Computer Vision-Part II, ECCV '02*, pages 325–339, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-43744-4. URL <http://dl.acm.org/citation.cfm?id=645316.649195>.
- [43] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):603–619, may 2002. ISSN 0162-8828. doi: 10.1109/34.1000236.
- [44] Eric W Weisstein. Convolution. MathWorld—A Wolfram Web Resource, . URL <http://mathworld.wolfram.com/Convolution.html>.
- [45] Timo Ojala, Matti Pietika Inen, Senior Member, and Topi Maenpa A. Multiresolution grayscale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.
- [46] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1-3):185–203, August 1981. ISSN 00043702. doi: 10.1016/0004-3702(81)90024-2. URL <http://dspace.mit.edu/handle/1721.1/6337>.

- [47] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th international joint conference on Artificial intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc. URL <http://dl.acm.org/citation.cfm?id=1623264.1623280>.
- [48] Y. Zinbi, Y. Chahir, and A. Elmoataz. Moving object segmentation; using optical flow with active contour model. In *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, pages 1 –5, april 2008. doi: 10.1109/ICTTA.2008.4530112.
- [49] H. Sekkati and A. Mitiche. Joint optical flow estimation, segmentation, and 3d interpretation with level sets. *Comput. Vis. Image Underst.*, 103(2):89–100, August 2006. ISSN 1077-3142. doi: 10.1016/j.cviu.2005.11.002. URL <http://dx.doi.org/10.1016/j.cviu.2005.11.002>.
- [50] S. Denman, V. Chandran, and S. Sridharan. Adaptive optical flow for person tracking. In *Digital Image Computing: Techniques and Applications, 2005. DICTA '05. Proceedings 2005*, page 8, dec. 2005. doi: 10.1109/DICTA.2005.11.
- [51] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.
- [52] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical report, Carnegie Mellon University, April 1991. URL <http://www.ces.clemson.edu/~stb/klt/tomasi-kanade-techreport-1991.pdf>.
- [53] Jun-Sik Kim, Myung Hwangbo, and T. Kanade. Realtime affine-photometric klt feature tracker on gpu in cuda framework. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 886 –893, 27 2009-oct. 4 2009. doi: 10.1109/ICCVW.2009.5457608.
- [54] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150 –1157 vol.2, 1999. doi: 10.1109/ICCV.1999.790410.

- [55] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 29:2007, 2007.
- [56] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *In ECCV*, pages 404–417, 2006.
- [57] Kiran Varanasi, Andrei Zaharescu, Edmond Boyer, and Radu Horaud. Temporal surface tracking using mesh evolution. In *Proceedings of the 10th European Conference on Computer Vision: Part II*, ECCV '08, pages 30–43, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-88685-3. doi: 10.1007/978-3-540-88688-4_3. URL http://dx.doi.org/10.1007/978-3-540-88688-4_3.
- [58] Michael Jones, Paul Viola, Paul Viola, Michael J. Jones, Daniel Snow, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. In *In ICCV*, pages 734–741, 2003.
- [59] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proc. IEEE Computer Society Conf. Computer Vision and Pattern Recognition CVPR 2001*, volume 1, 2001. doi: 10.1109/CVPR.2001.990517.
- [60] Zdenek Kalal, Jiri Matas, and Krystian Mikolajczyk. Online learning of robust object detectors during unstable tracking. In *In International Conference on Computer Vision*, 2009.
- [61] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *In CVPR*, pages 886–893, 2005.
- [62] T. Ojala, M. Pietikainen, and D. Harwood. Performance evaluation of texture measures with classification based on kullback discrimination of distributions. In *Pattern Recognition, 1994. Vol. 1 - Conference A: Computer Vision Image Processing., Proceedings of the 12th IAPR International Conference on*, volume 1, pages 582–585 vol.1, oct 1994. doi: 10.1109/ICPR.1994.576366.
- [63] Yadong Mu, Shuicheng Yan, Yi Liu, T. Huang, and Bingfeng Zhou. Discriminative local binary patterns for human detection in personal album. In *Computer Vision*

- and Pattern Recognition, 2008. *CVPR 2008. IEEE Conference on*, pages 1 –8, june 2008. doi: 10.1109/CVPR.2008.4587800.
- [64] Huadong Ma, Chengbin Zeng, and Charles X. Ling. A reliable people counting system via multiple cameras. *ACM Trans. Intell. Syst. Technol.*, 3(2):31:1–31:22, February 2012. ISSN 2157-6904. doi: 10.1145/2089094.2089107. URL <http://doi.acm.org/10.1145/2089094.2089107>.
- [65] John G. Daugman. Two-dimensional spectral analysis of cortical receptive field profiles. *Vision Research*, 20(10):847 – 856, 1980. ISSN 0042-6989. doi: 10.1016/0042-6989(80)90065-6. URL <http://www.sciencedirect.com/science/article/pii/0042698980900656>.
- [66] T. Serre, M. Kouh, C. Cadieu, U. Knoblich, G. Kreiman, T. Poggio, Thomas Serre, Minjoon Kouh, Charles Cadieu, Ulf Knoblich, Gabriel Kreiman, and Tomaso Poggio. A theory of object recognition: Computations and circuits in the feedforward path of the ventral stream in primate visual cortex. In *AI Memo*, 2005.
- [67] J. G. Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *Journal of the Optical Society of America A: Optics, Image Science, and Vision*, 2(7):1160–1169, 1985.
- [68] Chengjun Liu and Harry Wechsler. Gabor feature based classification using the enhanced fisher linear discriminant model for face recognition. *IEEE Trans. Image Processing*, 11:467–476, 2002.
- [69] K.B. Vinay and B.S. Shreyas. Face recognition using gabor wavelets. In *Signals, Systems and Computers, 2006. ACSSC '06. Fortieth Asilomar Conference on*, pages 593 –597, 29 2006-nov. 1 2006. doi: 10.1109/ACSSC.2006.354817.
- [70] David A. Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *Int. J. Comput. Vision*, 77(1-3):125–141, May 2008. ISSN 0920-5691. doi: 10.1007/s11263-007-0075-7. URL <http://dx.doi.org/10.1007/s11263-007-0075-7>.

- [71] A.D. Jepson, D.J. Fleet, and T.F. El-Maraghi. Robust online appearance models for visual tracking. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(10):1296 – 1311, oct. 2003. ISSN 0162-8828. doi: 10.1109/TPAMI.2003.1233903.
- [72] E.P. Simoncelli, W.T. Freeman, E.H. Adelson, and D.J. Heeger. Shiftable multiscale transforms. *Information Theory, IEEE Transactions on*, 38(2):587 –607, mar 1992. ISSN 0018-9448. doi: 10.1109/18.119725.
- [73] Shaohua Kevin Zhou, R. Chellappa, and B. Moghaddam. Visual tracking and recognition using appearance-adaptive models in particle filters. *Image Processing, IEEE Transactions on*, 13(11):1491 –1506, nov. 2004. ISSN 1057-7149. doi: 10.1109/TIP.2004.836152.
- [74] Raquel Urtasun. 3d people tracking with gaussian process dynamical models. In *CVPR*, pages 238–245. CVPR, 2006.
- [75] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 142 –149 vol.2, 2000. doi: 10.1109/CVPR.2000.854761.
- [76] Deva Ramanan, David A. Forsyth, and Andrew Zisserman. Tracking people by learning their appearance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(1):65–81, January 2007. ISSN 0162-8828. doi: 10.1109/TPAMI.2007.22. URL <http://dx.doi.org/10.1109/TPAMI.2007.22>.
- [77] Junliang Xing, Haizhou Ai, Liwei Liu, and Shihong Lao. Multiple player tracking in sports video: A dual-mode two-way bayesian inference approach with progressive observation modeling. *Image Processing, IEEE Transactions on*, 20(6):1652 –1667, june 2011. ISSN 1057-7149. doi: 10.1109/TIP.2010.2102045.
- [78] S.T. Birchfield and Sriram Rangarajan. Spatiograms versus histograms for region-based tracking. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 1158 – 1163 vol. 2, june 2005. doi: 10.1109/CVPR.2005.330.

- [79] C.O. Conaire, N.E. O'Connor, and A.F. Smeaton. An improved spatiogram similarity measure for robust object localisation. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 1, pages I-1069 –I-1072, april 2007. doi: 10.1109/ICASSP.2007.366096.
- [80] O.M. Lozano and K. Otsuka. Simultaneous and fast 3d tracking of multiple faces in video by gpu-based stream processing. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 713 –716, 31 2008-april 4 2008. doi: 10.1109/ICASSP.2008.4517709.
- [81] Jian Yao and Jean M. Odobez. *Multi-Person Bayesian Tracking with Multiple Cameras*, chapter 15, page 363. ELSEVIER, 2009.
- [82] J. Berclaz, F. Fleuret, E. Turetken, and P. Fua. Multiple object tracking using k-shortest paths optimization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(9):1806 –1819, sept. 2011. ISSN 0162-8828. doi: 10.1109/TPAMI.2011.21.
- [83] C. Stauffer and W. E. L. Grimson. Adaptive background mixture models for real time tracking. In *Proc. IEEE Computer Society Conference on. Computer Vision and Pattern Recognition*, volume 2, page 252 Vol. 2, 1999. doi: 10.1109/CVPR.1999.784637.
- [84] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: principles and practice of background maintenance. In *Proc. Seventh IEEE International Conference on Computer Vision The*, volume 1, page 255 261 vol.1, 1999. doi: 10.1109/ICCV.1999.791228.
- [85] F. Fleuret, J. Berclaz, R. Lengagne, and P. Fua. Multicamera people tracking with a probabilistic occupancy map. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(2):267 –282, feb. 2008. ISSN 0162-8828. doi: 10.1109/TPAMI.2007.1174.
- [86] Ben Benfold and Ian Reid. Stable multi-target tracking in real-time surveillance video. In *CVPR*, pages 3457–3464, 2011.

- [87] Xiaoyu Wang, Tony X. Han, and Shuicheng Yan. An hog-lbp human detector with partial occlusion handling. In *ICCV*, pages 32–39, 2009.
- [88] Chao He, Y.F. Zheng, and S.C. Ahalt. Object tracking using the gabor wavelet transform and the golden section algorithm. *Multimedia, IEEE Transactions on*, 4(4):528 – 538, dec 2002. ISSN 1520-9210. doi: 10.1109/TMM.2002.806534.
- [89] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.
- [90] Lili Ma, Jing Liu, Jinqiao Wang, Jian Cheng, and Hanqing Lu. A improved silhouette tracking approach integrating particle filter with graph cuts. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 1142 –1145, march 2010. doi: 10.1109/ICASSP.2010.5495366.
- [91] Mark Pupilli and Andrew Calway. Real-time camera tracking using a particle filter. In *In Proc. British Machine Vision Conference*, pages 519–528, 2005.
- [92] C. Garate, P. Bilinsky, and F. Bremond. Crowd event recognition using hog tracker. In *Performance Evaluation of Tracking and Surveillance (PETS-Winter), 2009 Twelfth IEEE International Workshop on*, pages 1 –6, dec. 2009. doi: 10.1109/PETS-WINTER.2009.5399727.
- [93] Fei Yan, Alexey Kostin, William Christmas, and Josef Kittler. A novel data association algorithm for object tracking in clutter with application to tennis video analysis. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1, CVPR '06*, pages 634–641, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2597-0. doi: 10.1109/CVPR.2006.36. URL <http://dx.doi.org/10.1109/CVPR.2006.36>.
- [94] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, April 1972. ISSN 0004-5411. doi: 10.1145/321694.321699. URL <http://doi.acm.org/10.1145/321694.321699>.

- [95] Harold W Kuhn. Variants of the hungarian method for assignment problems. *Naval Research Logistics Quarterly*, 3:253-258, 1956.
- [96] James Munkres. Algorithm for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, Vol.5 N0.1:32-38, 1957.
- [97] Stuart Russell and Peter Norving. *Artificial Intelligence a Modern Approach third edition*. Peason Education, 2010.
- [98] Grzegorz Cielniak and Tom Duckett. People recognition by mobile robots. *Journal of Intelligent and Fuzzy Systems*, Vol, 15:21-27, 2004.
- [99] Qiang Zhu, Shai Avidan, Mei chen Yeh, and Kwang ting Cheng. Fast human detection using a cascade of histograms of oriented gradients. In *In CVPR06*, pages 1491-1498, 2006.
- [100] V Prisacariu and I Reid. fasthog - a real-time gpu implementation of hog. Technical report, Department of Engineering Science, University of Oxford, 2009.
- [101] Jerome Berclaz, Ali Shahrokni, Francois Fleuret, James Ferryman, and Pascal Fua. Evaluation of probabilistic occupancy map people detection for surveillance systems. In *Proceedings of the IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*, 0 2009.
- [102] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, 21(1):32 - 40, jan 1975. ISSN 0018-9448. doi: 10.1109/TIT.1975.1055330.
- [103] Yizong Cheng. Mean shift, mode seeking, and clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(8):790 -799, aug 1995. ISSN 0162-8828. doi: 10.1109/34.400568.
- [104] S.J. Julier and J.K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401 - 422, mar 2004. ISSN 0018-9219. doi: 10.1109/JPROC.2003.823141.

- [105] Joon Woong Lee, Mun Sang Kim, and In So Kweon. A kalman filter based visual tracking algorithm for an object moving in 3d. In *Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, volume 1, pages 342–347 vol.1, aug 1995. doi: 10.1109/IROS.1995.525818.
- [106] Nathan Funk. A study of the kalman filter applied to visual tracking. Technical report, University of Alberta, 2003.
- [107] L. Bazzani, D. Bloisi, and V. Murino. A comparison of multi-hypothesis kalman filter and particle filter for multi-target tracking. In *Performance Evaluation of Tracking and Surveillance workshop at CVPR 2009*, pages 47–54, Miami, Florida, 2009.
- [108] Albert N. Shiryaev. *Essentials of Stochastic Finance: Facts, Models, Theory*. World Scientific Publishing Company, 1st edition, April 1999. ISBN 9810236050. URL <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/9810236050>.
- [109] Linda J. Allen. *An Introduction to Stochastic Processes with Biology Applications*. Prentice Hall, April 2003. ISBN 0130352187. URL <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0130352187>.
- [110] C. W. Helstrom. *Probability and Stochastic processes for Engineers*. Maxwell, Macmillan, New York, 2nd edition edition, 1991.
- [111] J Harris and H Stocker. *Handbook of Mathematics and Computational Science, p824*. New York: Springer-Verlag, 1998.
- [112] B Ristic, S Arulampalam, and N Gordon. *Beyond the Kalman filter Particle filter for Tracking Applications*. Artech House, 2004.
- [113] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44:335–341, 1949.

-
- [114] Eric W. Weisstein. Monte carlo integration, . URL <http://mathworld.wolfram.com/MonteCarloIntegration.html>.
- [115] Harvey Gould, Jan Tobochnik, and Christian Wolfgang. *An Introduction to Computer Simulation Methods: Applications to Physical Systems (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005. ISBN 0805377581.
- [116] John Geweke. Bayesian inference in econometric models using monte carlo integration. *Econometrica*, 57(6):1317–1339, 1989. doi: 10.2307/1913710. URL <http://dx.doi.org/10.2307/1913710>.
- [117] John von Neumann. Various techniques used in connection with random digits. *National Bureau of Standards, Applied Math Series*, 11:36–38, 1951.
- [118] L Devroye. *Non-Uniform Random Variate Generation*. New York: Springer-Verlag, 1986.
- [119] N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE Proceedings F Radar and Signal Processing*, 140(2):107–113, April 1993.
- [120] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, April 1970. ISSN 1464-3510. doi: 10.1093/biomet/57.1.97. URL <http://dx.doi.org/10.1093/biomet/57.1.97>.
- [121] Peter G. Harrison. Turning back time in markovian process algebra. *Theor. Comput. Sci.*, 290(3):1947–1986, January 2003. ISSN 0304-3975. doi: 10.1016/S0304-3975(02)00375-4. URL [http://dx.doi.org/10.1016/S0304-3975\(02\)00375-4](http://dx.doi.org/10.1016/S0304-3975(02)00375-4).
- [122] S Richardson and P Green. On bayesian analysis of mixtures with an unknown number of components. *Journal of the Royal Statistical Society Series B (Methodological)*, Vol. 59, No. 4, 1997.

- [123] S Marin and A Wallace. Multilayered 3d lidar image construction using spatial models in a bayesian framework. *Pattern Analysis and Machine Intelligence, IEEE Transactions*, 2008.
- [124] P Green. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 1995.
- [125] D Wilkinson. *Parallel Bayesian Computation, Handbook of Parallel Computing and Statistics*. Marcel Dekker/CPC Press, 2005.
- [126] P Solymos. Parallel computing with bayesian mcmc and data cloning in r with the dclone package. Technical report, University of Alberta, 2011.
- [127] Harald Niederreiter. *Random number generation and quasi-Monte Carlo methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992. ISBN 0-89871-295-5.
- [128] I Sobol. The distribution of points in a cube and the approximate evaluation of integrals. *U.S.S.R Comput. Maths. Math. Phys*, 7:86–112, 1967.
- [129] NVIDIA. Curand guide, 2012. PG-05328-041-v01.
- [130] R Shepard. Toward a universal law of generalization for psychological science. *Science*, 237:1317–1323, 1987.
- [131] A. Hartley, R. Zisserman. *Multiple View Geometry in computer vision, 2nd edition*. Cambridge University Press, 2003.
- [132] R. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal on Robotics and Automation*, 3(4):323–344, 1987. ISSN 0882-4967. doi: 10.1109/JRA.1987.1087109. URL <http://dx.doi.org/10.1109/JRA.1987.1087109>.
- [133] Z. Zhang. A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(11):1330 – 1334, nov 2000. ISSN 0162-8828. doi: 10.1109/34.888718.

- [134] Wende Zhang and Tsuhan Chen. A probabilistic framework for geometry reconstruction using prior information. In *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, volume 2, pages II –529 –II –532, 16 2007-oct. 19 2007. doi: 10.1109/ICIP.2007.4379209.
- [135] Feng Li, Jingyi Yu, and Jinxiang Chai. A hybrid camera for motion deblurring and depth map super-resolution. In *In CVPR 08: Proc. of the 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2008.
- [136] Jean Yves Bouguet. Camera calibration toolbox for matlab, July 2010. URL http://www.vision.caltech.edu/bouguetj/calib_doc/.
- [137] Brett R. Jones. Augmenting complex surfaces with projector-camera systems. Master’s thesis, University of Illinois at Urbana-Champaign, 2010.
- [138] Prasanna Sundararajan. High performance computing using fpgas. Technical report, Xilinx, 2010. URL http://www.xilinx.com/support/documentation/white_papers/wp375_HPC_Using_FPGAs.pdf.
- [139] Xilinx. 7 series fpgas overview. Technical report, October 2012. URL http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf.
- [140] Miguel Arias-Estrada and Eduardo Rodrguez-Palacios. An fpga co-processor for real-time visual tracking. In Manfred Glesner, Peter Zipf, and Michel Renovell, editors, *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, volume 2438 of *Lecture Notes in Computer Science*, pages 73–98. Springer Berlin / Heidelberg, 2002. ISBN 978-3-540-44108-3. URL http://dx.doi.org/10.1007/3-540-46117-5_73.
- [141] Jinbo Xu, Yong Dou, Junfeng Li, Xingming Zhou, and Qiang Dou. Fpga accelerating algorithms of active shape model in people tracking applications. In *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, DSD ’07*, pages 432–435, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2978-X. doi: 10.1109/DSD.2007.59. URL <http://dx.doi.org/10.1109/DSD.2007.59>.

- [142] Chun Hok Ho, Chi Wai Yu, P. Leong, W. Luk, and S. Wilton. Floating-point fpga: Architecture and modeling. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 17(12):1709–1718, dec. 2009. ISSN 1063-8210. doi: 10.1109/TVLSI.2008.2006616.
- [143] Nikolay Sorokin. Implementation of high-speed fixed-point dividers on fpga. 2006.
- [144] F. de Dinechin, M. Joldes, B. Pasca, and G. Revy. Multiplicative square root algorithms for fpgas. In *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, pages 574–577, 31 2010-sept. 2 2010. doi: 10.1109/FPL.2010.112.
- [145] Ray Andraka. A survey of cordic algorithms for fpga based computers. In *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays, FPGA '98*, pages 191–200, New York, NY, USA, 1998. ACM. ISBN 0-89791-978-5. doi: 10.1145/275107.275139. URL <http://doi.acm.org/10.1145/275107.275139>.
- [146] Xilinx. Virtex-6 family overview, 2012. URL http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf.
- [147] Xilinx. LogiCore ip floating-point operator v5.0, March 2011. URL http://www.xilinx.com/support/documentation/ip_documentation/floating_point_ds335.pdf.
- [148] Srinidhi Kestur, John D. Davis, and Oliver Williams. Blas comparison on fpga, cpu and gpu. In *Proceedings of the 2010 IEEE Annual Symposium on VLSI, ISVLSI '10*, pages 288–293, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4076-4. doi: 10.1109/ISVLSI.2010.84. URL <http://dx.doi.org/10.1109/ISVLSI.2010.84>.
- [149] David Barrie Thomas, Lee Howes, and Wayne Luk. A comparison of cpus, gpus, fpgas, and massively parallel processor arrays for random number generation. In *Proceedings of the ACM/SIGDA international symposium on Field programmable*

- gate arrays*, FPGA '09, pages 63–72, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-410-2. doi: 10.1145/1508128.1508139. URL <http://doi.acm.org/10.1145/1508128.1508139>.
- [150] Nvidia. Tesla c2050 / c2070 gpu computing processor supercomputing at 1/10th the cost, 2010. URL http://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_C2050_C2070_jul10_lores.pdf.
- [151] AMD. Amd firestream 9270 gpu compute accelerator, 2010. URL <http://www.amd.com/ru/products/server/processors/firestream/firestream-9270/pages/firestream-9270.aspx>.
- [152] Xilinx. LogiCore iP MicroBlaze micro controller system (v1.1), April 2012. URL http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ds865_microblaze_mcs.pdf.
- [153] A. Ellis, A. Shahrokni, and J.M. Ferryman. PETS2009 and winter-pets 2009 results: A combined evaluation. In *Performance Evaluation of Tracking and Surveillance (PETS-Winter), 2009 Twelfth IEEE International Workshop on*, pages 1–8, dec. 2009. doi: 10.1109/PETS-WINTER.2009.5399728.
- [154] D. Arsic, A. Lyutskanov, G. Rigoll, and B. Kwolek. Multi camera person tracking applying a graph-cuts based foreground segmentation in a homography framework. In *Proc. Twelfth IEEE Int Performance Evaluation of Tracking and Surveillance (PETS-Winter) Workshop*, pages 1–8, 2009.
- [155] Saad M. Khan and Mubarak Shah. A multiview approach to tracking people in crowded scenes using a planar homography constraint. In *In European Conference on Computer Vision*, 2006.
- [156] J. Berclaz, F. Fleuret, and P. Fua. Multiple object tracking using flow linear programming. In *Performance Evaluation of Tracking and Surveillance (PETS-Winter), 2009 Twelfth IEEE International Workshop on*, pages 1–8, dec. 2009. doi: 10.1109/PETS-WINTER.2009.5399488.

- [157] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool. Robust tracking-by-detection using a detector confidence particle filter. In *Proc. IEEE 12th Int Computer Vision Conf*, pages 1515–1522, 2009.
- [158] Z.L. Husz, A.M. Wallace, and P.R. Green. Tracking with a hierarchical partitioned particle filter and movement modelling. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 41(6):1571–1584, dec. 2011. ISSN 1083-4419. doi: 10.1109/TSMCB.2011.2157680.
- [159] Lawrence C. Evans. An introduction to stochastic differential equations – version 1.2. Technical report, Department of Mathematics, UC Berkeley, 2006.
- [160] J. Ferryman and A. Shahrokni. Pets2009: Dataset and challenge. In *Proc. Twelfth IEEE Int Performance Evaluation of Tracking and Surveillance (PETS-Winter) Workshop*, pages 1–6, 2009. doi: 10.1109/PETS-WINTER.2009.5399556.
- [161] Eric W Weisstein. Euler angles, . URL <http://mathworld.wolfram.com/EulerAngles.html>.
- [162] D Eberly. Perspective projection of an ellipse. 1999. www.geometrictools.com.
- [163] Eric Weisstein. Sphere point picking. From MathWorld—A Wolfram Web Resource, 1999. URL <http://mathworld.wolfram.com/SpherePointPicking.html>.
- [164] Philip J. Schneider and David H. Eberly. *Geometric Tools for Computer Graphics*. Elsevier, 2003.
- [165] Ouguz Karan, Canan Bayraktar, Haluk Gumuskaya, and Bekir Karlik. Diagnosing diabetes using neural networks on small mobile devices. *Expert Syst. Appl.*, 39(1): 54–60, January 2012. ISSN 0957-4174. doi: 10.1016/j.eswa.2011.06.046. URL <http://dx.doi.org/10.1016/j.eswa.2011.06.046>.
- [166] Richard Bellman. *Dynamic Programming*. Dover Publications, 2003. ISBN 0486428095.

- [167] Konstantinos G. Derpanis. Integral image-based representations. Technical report, Department of Computer Science and Engineering, York University, 2007. URL http://www.cse.yorku.ca/~kosta/CompVis_Notes/integral_representations.pdf.
- [168] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: the clear mot metrics. *J. Image Video Process.*, 2008:1–10, 2008. ISSN 1687-5176. doi: <http://dx.doi.org/10.1155/2008/246309>.
- [169] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM. doi: 10.1145/1465482.1465560. URL <http://doi.acm.org/10.1145/1465482.1465560>.
- [170] Nvidia. *CUDA C Programming Guide*. Nvidia, version 3.2 edition, 7 2010.
- [171] Nvidia. *CUDA C Best Practice Guide*. Nvidia, version 3.1 edition, 5 2010.
- [172] Michael J. Quinn. *Parallel Computing Theory and Practice*. McGraw-Hill International Editions, 1994.
- [173] Michael J. Flynn and Kevin W. Rudd. Parallel architectures. *ACM Comput. Surv.*, 28(1):67–70, March 1996. ISSN 0360-0300. doi: 10.1145/234313.234345. URL <http://doi.acm.org/10.1145/234313.234345>.
- [174] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492. URL <http://doi.acm.org/10.1145/1327452.1327492>.
- [175] P. W. Trinder, M. I. Cole, H w. Loidl, and G. J. Michaelson. Characterising effective resource analyses for parallel and distributed coordination. Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.1898, 2011.

- [176] Mark Harris. Optimizing parallel reduction in cuda, 2008. URL <http://people.maths.ox.ac.uk/gilesm/cuda/prac4/reduction.pdf>.
- [177] Harald Niederreiter. Low-discrepancy and low-dispersion sequences. *Journal of Number Theory*, 30(1):51 – 70, 1988. ISSN 0022-314X. doi: 10.1016/0022-314X(88)90025-X. URL <http://www.sciencedirect.com/science/article/pii/0022314X8890025X>.
- [178] Paul Bratley and Bennett L. Fox. Algorithm 659: Implementing sobol’s quasirandom sequence generator. *ACM Trans. Math. Softw.*, 14(1):88–100, March 1988. ISSN 0098-3500. doi: 10.1145/42288.214372. URL <http://doi.acm.org/10.1145/42288.214372>.
- [179] Jason Sanders and Edward Kandrot. *CUDA by example : an introduction to general-purpose GPU programming*. Addison Wesley, 2010.
- [180] Jinman Kang, Isaac Cohen, and Gerard Medioni. Persistent objects tracking across multiple non overlapping cameras. In *Application of Computer Vision, 2005. WACV/MOTIONS ’05 Volume 1. Seventh IEEE Workshops on*, volume 2, pages 112 –119, jan. 2005. doi: 10.1109/ACVMOT.2005.92.
- [181] Guoyun Lian, Jian-Huang Lai, C.Y. Suen, and Pei Chen. Matching of tracked pedestrians across disjoint camera views using ci-dlbp. *Circuits and Systems for Video Technology, IEEE Transactions on*, 22(7):1087 –1099, july 2012. ISSN 1051-8215. doi: 10.1109/TCSVT.2012.2190471.
- [182] O. Javed, Z. Rasheed, K. Shafique, and M. Shah. Tracking across multiple cameras with disjoint views. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 952 –957 vol.2, oct. 2003. doi: 10.1109/ICCV.2003.1238451.
- [183] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. URL <http://scitation.aip.org/getabs/servlet/GetabsServlet?prog=normal&id=SMJCAT000002000004000225000001&idtype=cvips&gifs=yes>.

- [184] C. Madden and M. Piccardi. A framework for track matching across disjoint cameras using robust shape and appearance features. In *Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on*, pages 188 – 193, sept. 2007. doi: 10.1109/AVSS.2007.4425308.
- [185] L. Lo Presti, S. Sclaroff, and M.L. Cascia. Path modeling and retrieval in distributed video surveillance databases. *Multimedia, IEEE Transactions on*, 14(2): 346 –360, april 2012. ISSN 1520-9210. doi: 10.1109/TMM.2011.2173323.
- [186] Eric W. Weisstein. Ramp function, . URL <http://mathworld.wolfram.com/RampFunction.html>. retrieved August 2012.
- [187] John Makhoul, Francis Kubala, Richard Schwartz, and Ralph Weischedel. Performance measures for information extraction. In *In Proceedings of DARPA Broadcast News Workshop*, pages 249–252, 1999.
- [188] K. Madsen, H. B. Nielsen, and O. Tingleff. Methods for non-linear least squares problems 2nd edition. Informations and Mathematical modeling, Tecnical University of Denmark, 2004.
- [189] P. Mittrapiyanuruk. A memo on how to use the levenberg-marquardt algorithm for refining camera calibration parameters. Purdue University, 2006. URL https://engineering.purdue.edu/kak/courses-i-teach/ECE661/HW5_LM_handout.pdf.